Appendix "O"

FILTER

```
#include <stdio.h>
#include <signal.h>
#include <ctype.h>
#include <unistd.h>
#include <string.h>
#include <sys/stat.h>
#include <math.h>
#include "parseopt.h"
#include "utl_str.h"
#include "utl_mem.h"
#include "utl_file.h"
#include "utl_math.h"
#include "ct.h"
#include "ct_expr.h"
#include "ct_proto.h"
#include "import_proto.h"
#include "io_fprint.h"

#include "dservTypes.h"

/*
**+E:
**
**
** Abstract      : Function parses range field string for ADS design programs.
**                 It takes a string of the form
**                 "logp -1.0 8.0 MW 200 500 price 0 12.50" and fills in the
**                 global array  RangeFields.
**
**
```

```
    ** Usage         :
    **
    ** Returns       : 1 on success, 0 for failure.
    **
 5  ** Algorithms    : None.
    **
    **
    **
    **-E:
10  */
    int ParseRangeVar(rangeVar,numRangeFieldsAllocated,numRangeFields,rangeFields)
    char   *rangeVar ;
    int    *numRangeFieldsAllocated ;
    int    *numRangeFields ;
15  struct RangeStruct **rangeFields;
    {
    static int stat = 0 ;
    char *buffer = (char *)NULL ;
    char *name ;
20  char *low ;
    char *high ;
    int  i ;

        *numRangeFieldsAllocated = 0 ;
        *numRangeFields = 0 ;
25      *rangeFields = (struct RangeStruct *)NULL ;

            if ( !(buffer = UTL_STR_SAVE(rangeVar)) )
                    goto Failure ;

        name = strtok(buffer," ");
        while ( name )
30      {
```

```
        if ( !(low  = strtok(NULL," ")) )
                        goto UnableToParse ;
        if ( !(high = strtok(NULL," ")) )
                        goto UnableToParse ;
5       if ( *numRangeFields >= *numRangeFieldsAllocated )
        {
                if ( !*rangeFields )
                {
                                if (!(*rangeFields = (struct RangeStruct
10  *)UTL_MEM_CALLOC(

    ALLOCATE_INCREMENT,
                                                sizeof(struct RangeStruct))))
                                        goto Failure ;
15                              else
                                        *numRangeFieldsAllocated =
    ALLOCATE_INCREMENT ;
                }
                        else
                        {
20                              if (!( *rangeFields = (struct RangeStruct
    *)UTL_MEM_RECALLOC(
                                                rangeFields,
                        (*numRangeFieldsAllocated *sizeof(struct RangeStruct)),
25                      ((*numRangeFieldsAllocated + ALLOCATE_INCREMENT) *
                           sizeof(struct RangeStruct)) )) )
                                        goto Failure ;
                                else
                                        *numRangeFieldsAllocated += 
30  ALLOCATE_INCREMENT ;

                        }
        }
```

```
                (*rangeFields)[*numRangeFields].RangeFieldName =
UTL_STR_SAVE(name);
                (*rangeFields)[*numRangeFields].lowValue = atof(low);
                (*rangeFields)[*numRangeFields].highValue = atof(high);
        (*numRangeFields)++ ;
                name = strtok(NULL," ");
    }
    stat = 1 ;
    goto CleanUp ;

UnableToParse:
        fprintf(stderr,"Unable to parse -rangevar %s\n",rangeVar);
        stat = 0 ;
    goto CleanUp ;
Failure :
    stat = 0 ;
    goto CleanUp ;
CleanUp :
        if ( buffer )
                UTL_MEM_FREE(buffer);
    return stat ;
}

/*
**+E:
**
**
** Abstract      : Function parses one of field string for ADS design programs.
**                 It takes a string of the form
**                 "supplier Aldrich,Sigma,Fluka,SALOR taste SWEET,Salty"
**                 global array  OneOfValues.
**
**
```

```
** Usage         :
**
** Returns       : 1 on success, 0 for failure.
**
** Algorithms    : None.
**
**
**
**-E:
*/
int ParseOneOfVar(oneOfVar,numOneOfFieldsAllocated,numOneOfFields,oneOfValues)
char   *oneOfVar ;
int    *numOneOfFieldsAllocated ;
int    *numOneOfFields ;
struct OneOfStruct **oneOfValues;
{
static int stat = 0 ;
char *buffer = (char *)NULL ;
char *name ;
char *choices ;
char *choice ;
int  i ;
int  j ;
char *cp ;
char *end ;
    *numOneOfFieldsAllocated = 0 ;
    *numOneOfFields = 0 ;
    (*oneOfValues) = (struct OneOfStruct *)NULL ;

        if ( !(buffer = UTL_STR_SAVE(oneOfVar)) )
                goto Failure ;
 /*
 ** Start off by reading the field name .
```

```
*/
    name = strtok(buffer," ");
    while ( name )
    {
        if ( *numOneOfFields >= *numOneOfFieldsAllocated )
        {
            if ( !(*oneOfValues) )
            {
                if (!(*oneOfValues = (struct OneOfStruct
*)UTL_MEM_CALLOC(
ALLOCATE_INCREMENT,
                        sizeof(struct OneOfStruct))))
                    goto Failure ;
                else
                    *numOneOfFieldsAllocated =
ALLOCATE_INCREMENT ;
            }
            else
            {
                if (!( *oneOfValues = (struct OneOfStruct
*)UTL_MEM_RECALLOC(
                        *oneOfValues,
                (*numOneOfFieldsAllocated *sizeof(struct OneOfStruct)),
                ((*numOneOfFieldsAllocated + ALLOCATE_INCREMENT) *
                    sizeof(struct OneOfStruct)) )) )
                    goto Failure ;
                else
                    *numOneOfFieldsAllocated +=
ALLOCATE_INCREMENT ;

            }
        }
```

```
            (*oneOfValues)[*numOneOfFields].OneOfFieldName =
UTL_STR_SAVE(name);
            (*oneOfValues)[*numOneOfFields].numValues = 0 ;
            (*oneOfValues)[*numOneOfFields].numValuesAlloc =
ALLOCATE_INCREMENT ;
            if ( !((*oneOfValues)[*numOneOfFields].values = (char **)

UTL_MEM_CALLOC(ALLOCATE_INCREMENT,

sizeof(char *)) ) )
                goto Failure ;
/*
** Now look at the choices this field could have.
*/
            choices = strtok(NULL," ");
            if ( !choices )
                goto UnableToParse ;
            choice  = strtok(choices,",");
            while ( choice )
            {
                if ( (*oneOfValues)[*numOneOfFields].numValues >=
                        (*oneOfValues)[*numOneOfFields].numValuesAlloc )
                {
                        if ( !((*oneOfValues)[*numOneOfFields].values = (char **)

UTL_MEM_RECALLOC((*oneOfValues)[*numOneOfFields].values,
                                                (
(*oneOfValues)[*numOneOfFields].numValuesAlloc *
                            sizeof(char *)),
                          ( ((*oneOfValues)[*numOneOfFields].numValuesAlloc +
                                                    ALLOCATE_INCREMENT )
*sizeof(char *)) ) ))
                                    goto Failure ;
```

```
                        (*oneOfValues)[*numOneOfFields].numValuesAlloc+=
ALLOCATE_INCREMENT;
                    }

(*oneOfValues)[*numOneOfFields].values[(*oneOfValues)[*numOneOfFields].numValues]
 5  = UTL_STR_SAVE(choice);
                    (*oneOfValues)[*numOneOfFields].numValues++ ;
                    end = choice + strlen(choice) + 1 ;
                    choice = strtok(NULL,",");
        }
10              (*numOneOfFields)++;
                name = strtok(end," ");
    }

    stat = 1 ;
    goto CleanUp ;

15  UnableToParse:
        fprintf(stderr,"Unable to parse -oneof %s\n",oneOfVar);
        stat = 0 ;
    goto CleanUp ;
Failure :
20      stat = 0 ;
    goto CleanUp ;
CleanUp :
        if ( buffer )
            UTL_MEM_FREE(buffer);
25      return stat ;
}


/*
**+E:
```

```
**
**
** Abstract      : Function parses a line from the input file and extracts
**                out any rangevar or oneof fields.
**
**
** Usage         :
**
** Returns       : Always returns 1 ;
**
** Algorithms    : None.
**
**
**
**-E:
*/
int ReadLineAttributes(line,numRangeFields,rangeValues,rangeFields,numOneOfFields,
oneOfValues,oneOfFields)
char   *line ;
int    numRangeFields ;
float  **rangeValues;
struct RangeStruct *rangeFields;
int    numOneOfFields ;
int    **oneOfValues;
struct OneOfStruct *oneOfFields;
{
   int i ;
   int j ;
   char *cp ;
/*
** Now read in the salar selection fields if any.
*/
        if ( numRangeFields )
```

```
        {
                if ( !(*rangeValues = (float *)UTL_MEM_CALLOC(numRangeFields,

    sizeof(float)) ) )
5               return 0 ;
        }
        if ( numOneOfFields )
        {
                if ( !(*oneOfValues = (int *)UTL_MEM_CALLOC(numOneOfFields,
10
    sizeof(int)) ) )
                return 0 ;
        }
        for ( i = 0 ; i < numRangeFields ; i++ )
15      {
                if ( ( cp = strstr(line,rangeFields[i].RangeFieldName ) ) )
                {
    /*
    ** Move past the logp= to get the value of this field, if the value is
20  ** a ';' then it is a missing value.
    */
                        cp = cp + strlen(rangeFields[i].RangeFieldName) + 1 ;
                        if ( *cp == ';' )
                                (*rangeValues)[i] = MISSING_FLOAT_VALUE ;
25                      else
                                (*rangeValues)[i] = atof(cp);
                }
                else
                {
30                      (*rangeValues)[i] = MISSING_FLOAT_VALUE ;
                }
        }
    /*
```

```
    ** Parse the -oneof field, we are looking for something looking like
    ** "supplier=Aldrich"
    */
        for ( i = 0 ; i < numOneOfFields ; i++ )
        {
            if ( ( cp = strstr(line,oneOfFields[i].OneOfFieldName ) ) )
            {
                cp = cp + strlen(oneOfFields[i].OneOfFieldName) + 1 ;
                if ( *cp == ';' )
                    (*oneOfValues)[i] = MISSING_INT_VALUE ;
                else
                {
                    for ( j = 0 ; j < oneOfFields[i].numValues ; j++ )
                    {
                        if ( UTL_STR_NCMP_NOCASE(cp,
                        oneOfFields[i].values[j],
                        strlen(oneOfFields[i].values[j])) == 0)
                        {
                            (*oneOfValues)[i] = j ;
                            break;
                        }
                    }
                    if ( j == oneOfFields[i].numValues )
                        (*oneOfValues)[i] = NOT_A_MATCH_VALUE ;
                }

            }
            else
                (*oneOfValues)[i] = MISSING_INT_VALUE ;
        }
    }

    /*
```

```
**+E:
**
**
** Abstract      : Function Checks to see if the given product passes the
**                user supplied filters.
**
**
** Usage         :
**
** Returns       : 1 if the product is not within range, 0 otherwise.
**
** Algorithms    : None.
**
**
**-E:
*/
static int
NotWithinScalarRange(firstIndex,secondIndex,numRangeFields,rangeValues_Y01,rangeVal
ues_Y02,rangeFields,numOneOfFields,oneOfValues_Y01,oneOfValues_Y02,oneOfValues)
int firstIndex ;  /* Index into Y_01 data */
int secondIndex ; /* Index into Y_02 data */
int numRangeFields ;
float **rangeValues_Y01 ;
float **rangeValues_Y02 ;
struct RangeStruct *rangeFields;
int   numOneOfFields    ;
int   **oneOfValues_Y01 ;
int   **oneOfValues_Y02 ;
struct OneOfStruct *oneOfValues ;
{
int i ;
float total ;
```

```
    /*
    ** First check the range values.
    */
            for ( i = 0 ; i < numRangeFields ; i++ )
            {
    /*
    ** If one of the regions has a missing value, then we do not filter this
    ** product.
    */
                    if ((( rangeValues_Y01[firstIndex][i] - MISSING_FLOAT_VALUE )
                        == SMALL_FLOAT ) ||
                        (( rangeValues_Y02[secondIndex][i] - MISSING_FLOAT_VALUE )
                        == SMALL_FLOAT ) )
                    return 0 ;

                    total=rangeValues_Y01[firstIndex][i] + rangeValues_Y02[secondIndex][i];
                    if ((total > rangeFields[i].highValue ) ||
                            (total < rangeFields[i].lowValue ) )
                    {
                            return 1 ;
                    }
            }
            for ( i = 0 ; i < numOneOfFields ; i++ )
            {
    /*
    ** If the value is missing then we dont mess with this guy.
    */
                    if ( ( oneOfValues_Y01[firstIndex][i] == MISSING_INT_VALUE ) ||
                       ( oneOfValues_Y02[secondIndex][i] == MISSING_INT_VALUE ) )
                    return 0 ;
    /*
    ** If any of the regions in the product does not match the selection
    ** criteria, then the product is rejected.
```

```
    */
            if ( ( oneOfValues_Y01[firstIndex][i] == NOT_A_MATCH_VALUE ) ||
                (oneOfValues_Y02[secondIndex][i] == NOT_A_MATCH_VALUE ) )
                    return 1 ;
5       }

        return 0 ;
    }

    /*
    **+E:
10  **
    **
    ** Abstract      : Function zapps products who are not within the user supplied
    **                selection criteria.
    **
15  **
    ** Usage         :
    **
    ** Returns       : 1 on success or 0 on failure.
    **
20  ** Algorithms    : None.
    **
    **
    **-E:
    */
25  int FilterProducts(inputInfo,rangeData,oneOfData,numFiltered)
    struct InputInfoStruct *inputInfo ;
    struct RangeInfoStruct *rangeData ;
    struct OneOfInfoStruct *oneOfData ;
    int                    *numFiltered ;
30  {
    int numProducts ;
```

```
    int i ;
    int index1 ;
    int index2 ;
    int Y01_Offset = 0 ;
    int Y02_Offset = 0 ;
    int k ;

        *numFiltered = 0 ;


        for ( k = 0 ; k < inputInfo->totalInputs ; k++ )
        {
            numProducts = inputInfo->Y_01_Length[k] * inputInfo->Y_02_Length[k]
    ;
            for ( i = 0 ; i < numProducts ; i++ )
            {
                index1 = i / inputInfo->Y_02_Length[k]  ; /*Y_01 index */
                index2 = i % inputInfo->Y_02_Length[k]  ; /*Y_02 index */

                if ( NotWithinScalarRange(index1,
                    index2,
                    rangeData->numRangeFields ,
                    rangeData->rangeValues_Y01 + Y01_Offset ,
                    rangeData->rangeValues_Y02 + Y02_Offset ,
                    rangeData->rangeFields,
                    oneOfData->numOneOfFields    ,
                    oneOfData->oneOfValues_Y01 + Y01_Offset ,
                    oneOfData->oneOfValues_Y02 + Y02_Offset ,
                    oneOfData->oneOfFields ))
                {
                    ZapInputProduct(k,i,0,0);
    /*              FlagProduct(dead_Products,0,0,i  + bitMapStartPoint[k]  ); */
    /*              someLeft--;
```

```
                    remainingInput[k]-- ;  */
                        *numFiltered += 1 ;
                    }
                }
                Y01_Offset += inputInfo->Y_01_Length[k] ;
                Y02_Offset += inputInfo->Y_02_Length[k] ;
            }
            return 1 ;
        }
```

Appendix "R"

```
/*+M: dbcsln_bitset.c
*************************************************************************
**
**
** Author              Date          Description
** ====================  ============
=========================
** Fred Soltanshahi     07-30-96      Routines to access a ChemSpace
**                                    bitset.
** Entry Points :
**      CS_PRDCT_BITSET_OPEN() -- Opens a CS bitset file.
**      CS_PRDCT_BITSET_CLOSE()-- Closes and cleansup after a bitset file.
**      CS_PRDCT_BITSET_WRITE()-- Writes a bitset file.
**      CS_PRDCT_BITSET_CREATE()-- Creates a bitset in memory.
**      CS_PRDCT_BITSET_DUMP() -- Dumps the content of the file.
**      CS_PRDCT_BITSET_GETHITS-- Returns the indexes for the requested
**      number of hits.
**      CS_PRDCT_BITSET_SETBITS() -- Copies a raw bitset into ChemSpace
**                                   compressed bitset.
**      CS_PRDCT_BITSET_INDEXES_TO_INDEX -- get bitset index from
**                                   Y_01,Y_02 etc.
**      CS_PRDCT_BITSET_TO_RAW()  -- Copies a ChemSpace bitset back into
**                                   a raw bitset
**      CS_PRDCT_BITSET_CONCAT_RAW() -- Copies a ChemSpace bitset
**                                   into part of a raw bitset
**      CS_PRDCT_BITSET_SELECTED() -- returns totalSelected
**      CS_PRDCT_BITSET_REVEAL()  -- Returns elements of hidden bitset
**                                   data structure to external program.
**      CS_PRDCT_BITSET_SET_PRD_BIT() -- Sets a bit in ChemSpace bitset.
**      CS_PRDCT_BITSET_GET_STATS()  -- Returns totals for the bitset.
**      CS_PRDCT_BITSET_CORE_INFO()  -- Gets core information from a bitset
```

```
file.
**          CS_PRDCT_MSTR_CORE_INFO()  -- Gets core information from a master file.
**          CS_PRDCT_BITSET_GETHITS() --  Gets the indexes for the requested set of
**                                        hits.
**          CS_PRDCT_BITSET_CREATE_BIT_STRING() -- Creates a compressed
bitstring in memory
**          CS_PRDCT_BITSET_DESTROY_BIT_STRING()-- Deletes a compressed bitset
in memory.
**
*****************************************************************************
**
-M*/

#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <math.h>
#include "utl_str.h"
#include "utl_mem.h"
extern char *GetFullPathName();
extern char *basename();

#define VERSION_NUMBER 1.001
#define ALLOCATION_FACTOR 1.25  /*Extra room in each variation site for growth*/
#define MAX_VARIATION_SITES 255

extern int    setbits[8] ;
extern int    nbits[256] ;

typedef struct MasterFileStruct
{
        char *masterFilePathName ;
        int  masterRecNo ;
```

```
        char *prefixForFiles;
        int  numVariationSites ;
        int  numberOfMissingBits;
        int  lbits;
        char *corefilePathName;
        int  startCore;
        char *fingerFileName;
        int  fingerOffset ;
        char **x_FileName ;
        char **reagentInfo ;
} MasterFileStruct ;
typedef struct ProgramInfoStruct
{
        char *programName ;
        int  bufferSize ;
        int  *buffer ;
} ProgramInfoStruct ;
typedef struct BitSetFileStruct
{
        struct MasterFileStruct masterFileInfo ;
        struct ProgramInfoStruct programInfo ;
        int    numVariationSites ;
        int    *actuallSizes ;
        int    *allocSizes ;
        int    *numFragsInEachSite ;
        int    totalSelected ;
        int    *bitset ;
        int    firstHitAddress ;
    unsigned char **fragmentBitset;
} BitSetFileStruct ;
typedef struct
{
        struct BitSetFileStruct *bitset;
```

```
        int numVariations;
        int *choices;
        void *call_udata;              /* callback callback's udata */
        int (*funcptr)(void *udata, int numVariants, int *choices );
        int totalExcluded;
} BIT_TRACKING;
static int RangeCallback ( void *udata, int startRange, int endRange );

static int BitSetAddressToIndexes(struct BitSetFileStruct *bitset,int address,int **indxs,int
*notUsedFlag)
{
int    numVariations;
int *allPtr;           /* alloced Sizes Ptr */
int *actPtr;           /* actual sizes Ptr */
int *choices;
int *chPtr ;
int x ;
int i ;
int skip = 1;
int skipcnt = 0;
        numVariations = bitset->numVariationSites;
    x = address ;
/*
** If the caller did not give us space to put in the indexes, allocate our
** own.
*/
        if ( ! (*indxs) )
                (*indxs) = (int *) UTL_MEM_CALLOC(numVariations, sizeof(int) );
        choices = *indxs ;
        if ( notUsedFlag)
                *notUsedFlag = 0 ;
        for ( allPtr = bitset->allocSizes + (numVariations - 1),
                                        actPtr = bitset->actuallSizes + (numVariations
```

```
    - 1 ),
                                        chPtr = choices + (numVariations - 1 ),
                                        i = numVariations;

                        i-- ;
                        allPtr--,  actPtr--, chPtr-- )
5
        {
            *chPtr = x % *allPtr;
            if ( *chPtr >= *actPtr )
            {
                if ( !skipcnt )
10              {
                    skipcnt = 1;
                    skip *= ( *allPtr - *chPtr );
                }

15  /*
    ** If we are rejecting things out of bounds of the actual sizes then we
    ** are out of here.
    */
                    if ( notUsedFlag )
                    {
20                      *notUsedFlag = 1 ;
                        break;
                    }
                }
                x = x / *allPtr;
25              if ( !skipcnt )
                    skip *= *allPtr;
            }
            if ( !skipcnt )
                skip = 0;
30          return skip;
        }
```

```
static int TestDump(DumpCode, Parameter, DumpInt)
int DumpCode ;
void *Parameter ;
int  DumpInt ;
{
FILE *File = (FILE *)Parameter ;
        fprintf(File, "%d\n", DumpInt);
}


static int CalculateAllocatedSizes(int numSites,int *sizes,int *allocSizes)
{
int i ;
        for ( i = 0 ; i < numSites ; i++ )
                allocSizes[i] = sizes[i] * ALLOCATION_FACTOR ;
}

static int Init()
{
static int    firstTime = 1 ;
int    i ;
        if ( firstTime )
        {
                for (i=0;i<8;i++) setbits[i] = ( 1 << i) & 255;
                firstTime = 0 ;
                for (i=0;i<256;i++) nbits[i] = (i&1) + (i&2)/2 + (i&4)/4 + (i&8)/8 +
                                                        (i&16)/16 + (i&32)/32
  + (i&64)/64 + (i&128)/128 ;
        }
}

int setbits_nbits_Init()
{
```

```
    Init();
    return 1;
}


void *CreateCompressedBitSet(bitset,offset,numVariations,sizes,allocSizes)
int  *bitset ;
int  offset ;
int  numVariations ;
int  *sizes ;
int  *allocSizes ;
{
void *compressed ;
int    byte ;
int    bit ;
int    size ;
int    total ;
int    i ;
int    index1 ;
int    index2 ;
int    newPos ;
int    rowLength ;
unsigned char *bs = (unsigned char *)bitset ;
        Init();


#if 0
/*
** Always calculate the allocated sizes, we are the only one who can set this.
*/
        if ( allocSizes[0] <= 0 )
                CalculateAllocatedSizes(numVariations,sizes,allocSizes);
#endif
/*
```

```
** Make the allocated size the same as the real size.
*/
        for ( i = 0 ; i < numVariations ; i++ )
                allocSizes[i] = sizes[i] ;
        size = allocSizes[0] ;
        for ( i = 1 ; i < numVariations ; i++ )
                size *= allocSizes[i] ;
        total = sizes[0] ;
        for ( i = 1 ; i < numVariations ; i++ )
                total *= sizes[i] ;
        if ( sizes[0] != allocSizes[0] )
        {
                if ( !( compressed = (void *)IHBDeclare( size ) ) )
                        goto AddTraceback ;
/*
** Set the bitset if the caller supplied us with one.
*/
                if ( bitset )
                {
                        rowLength = sizes[1] ;
                        for ( i = 0 ; i < total ; i++ )
                        {
                                index1 = i / rowLength ;
                                index2 = i % rowLength ;
                                byte = ( i + offset ) / 8 ;
                                bit  = ( i + offset ) % 8 ;
                                if ( bs[byte] & setbits[bit] )
                                {
                                        newPos =  ( index1 * allocSizes[1] ) + index2 ;
                                        IHBSet(compressed, newPos );
                                }
                        }
                        IHBOptimize(compressed);
```

```
                }
        }
            else
            {
                if ( bitset )
                {
                if ( !( compressed = (void *)IHBDeclareWithInit(offset, size , bs )))
                        goto AddTraceback ;
                }
                else
                {
                if ( !( compressed = (void *)IHBDeclare( size ) )  )
                        goto AddTraceback ;
                }
        }
            return compressed ;
    AddTraceback :
            return (void *)NULL ;
    }
    /*
    ** This routine will create a compressed bitset that is bigger in every
    ** dimension.
    */

    void *CreateCompressedBitSetExp(bitset,offset,numVariations,sizes,allocSizes)
    int  *bitset ;
    int  offset ;
    int  numVariations ;
    int  *sizes ;
    int  *allocSizes ;
    {
    void *compressed ;
    int    byte ;
```

```
    int    bit ;
    int    size ;
    int    total ;
    int    i ;
    int    index1 ;
    int    index2 ;
    int    newPos ;
    int    rowLength ;
    unsigned char *bs = (unsigned char *)bitset ;
        Init();
    /*
    ** Always calculate the allocated sizes, we are the only one who can set this.
    */
        if ( allocSizes[0] <= 0 )
            CalculateAllocatedSizes(numVariations,sizes,allocSizes);
        size = allocSizes[0] ;
        for ( i = 1 ; i < numVariations ; i++ )
            size *= allocSizes[i] ;
        total = sizes[0] ;
        for ( i = 1 ; i < numVariations ; i++ )
            total *= sizes[i] ;
        if ( sizes[0] != allocSizes[0] )
        {
            if ( !( compressed = (void *)IHBDeclare( size ) ) )
                goto AddTraceback ;
    /*
    ** Set the bitset if the caller supplied us with one.
    */
            if ( bitset )
            {
                rowLength = sizes[1] ;
                for ( i = 0 ; i < total ; i++ )
                {
```

```
                        index1 = i / rowLength ;
                        index2 = i % rowLength ;
                        byte = ( i + offset ) / 8 ;
                        bit  = ( i + offset ) % 8 ;
5                       if ( bs[byte] & setbits[bit] )
                        {
                                newPos =  ( index1 * allocSizes[1] ) + index2 ;
                                IHBSet(compressed, newPos );
                        }
10                  }
                }
        }
            else
            {
15              if ( bitset )
                {
                if ( !( compressed = (void *)IHBDeclareWithInit(offset, size , bs )))
                        goto AddTraceback ;
                }
20              else
                {
                if ( !( compressed = (void *)IHBDeclare( size ) )  )
                        goto AddTraceback ;
                }
25      }
            IHBOptimize(compressed);
            return compressed ;
        AddTraceback :
            return (void *)NULL ;
30  }
        CountBitSets(char *inputFileName)
        {
        FILE *inputFile ;
```

```
    int  i ;
    char *line ;
    long length ;
    long next ;
5   long here ;
        if ( !(inputFile = fopen(inputFileName,"r")))
                goto UnableToOpenFile ;
        i = 0 ;
        while ( 1 )
10      {
                here = ftell(inputFile);
                if (  -1 == UTL_SCAN_GETS( inputFile, "\\", "#", &line))
                        break;
                if ( UTL_STR_NCMP_NOCASE(line,"@BITSET_LENGTH:",15) != 0 )
15              {
    /*
    ** Kludge for dbsearch old format.
    */
                        if ( UTL_STR_NCMP_NOCASE(line,"@BITSET_START:",14) !=
20  0 )
                                break;
                }
                i++ ;
            length = (long ) atoi(line+15) ;
25          next = here + length  ;
                fseek(inputFile,next,SEEK_SET);
        }
        return i ;
    UnableToOpenFile :
30  AddTraceback :
        return 0 ;
    }
```

```
/*
**+E:
**
**
** Function Name : WriteOutCompressedBSFile()
**
** Purpose      : Function will write out a compressed bitset to disk.
**
** Usage :        To be used by dbsearch/design programs to write out check
**               points.
**
** Returns      : 1 on success, 0 for failure.
**
** Algorithms   : None.
**
** Revision History :
**
** Author                  Date        Description
** ======================      ========
==================
** Fred Soltanshahi        07/25/96    Original version.
**
**-E:
*/
int
WriteOutCompressedBSFile(outputFileName,masterFileName,masterRec,programName,co
mpressed,numVariations,sizes,allocSizes,numSelected,numInSites,bufferSize,buffer)
char *outputFileName ; /* Name of output file */
char *masterFileName ; /* Name of master file */
int  masterRec ;       /* Which record in the master file */
char *programName ;    /* Name of program generating this. */
void   *compressed ;   /* Compressed bitset */
int  numVariations ;   /* Num X01, X02 ... variation sites */
```

```
    int  *sizes ;          /* Actuall sizes in each dimension */
    int  *allocSizes ;     /* Allocated sizes(bitset size) in each dimension */
    int  numSelected ;      /* Number of products selected */
    int  *numInSites ;      /* Number of selections in each Y_0? site */
    int  bufferSize  ;      /* Program specific buffer size */
    int  *buffer     ;      /* Arbitrary data written by the program */
    {
    FILE   *outputFile ;
    float  version = VERSION_NUMBER ;
    long   bitsetSize ;
    time_t startTime ;
    int    i ;
    int    numBytes ;
    long bitsetStart = 0 ;
    long endOfFile   = 0 ;
    long length ;
    long beginingOfFile = 0 ;
    char *dir ;
    char *base ;
    char *fullPathName ;
    /*
    ** Calcualte the total size of the bitset.
    */
          bitsetSize = sizes[0] ;
          for ( i = 1 ; i < numVariations ; i++ )
                bitsetSize  = bitsetSize * sizes[i] ;
          if ( !(outputFile = fopen(outputFileName,"w")))
                goto UnableToOpenFile ;
     /*
    ** File format is :
    ** Version Number
    ** Date/Time Stamp
    ** "Location of the master file"  "Record Number in the file".
```

```
** A copy of master file records(one line at a time)
** Number of Variation sites
** Actual(current) number of choices for each site.
** Allocated number of choices for each site.
** Source -- program that generated this file.i.e. dbsearch, dbcsln_des,etc
** program command line parameters -- ASCII representation of parameters
** Program specific info          -- ASCII line of info specific to program.
** Number of products selected.
** Number of selections in each dimension -- num_Y_01_choices num_y_02_choices ...
** Bitset size -- The compressed bitset size in bytes.
** Bitset -- Row major bitset of products selected.
*/
/*      fprintf(outputFile,"@NEXT_SLOT:%010ld\n",endOfFile); */
        beginingOfFile = ftell(outputFile);
        fprintf(outputFile,"@BITSET_LENGTH:%010ld\n",bitsetStart);
        fprintf(outputFile,"@VERSION:%f\n",version);
        time( &startTime );
        fprintf(outputFile,"%s",ctime(&startTime));
        dir = GetFullPathName(masterFileName);
        base = basename(masterFileName,NULL);
        fullPathName = UTL_FILE_ADD_DIR_TO_DIRSPEC(dir,base);
        fprintf(outputFile,"%s %d\n",fullPathName,masterRec);
        UTL_MEM_FREE(fullPathName);
        if ( !DumpMasterFileInfo(outputFile,masterFileName,masterRec))
                goto UnableToDumpMaster ;
        fprintf(outputFile,"%d\n",numVariations);
        for ( i = 0 ; i < numVariations ; i++ )
                fprintf(outputFile,"%d ",sizes[i]);
        fprintf(outputFile,"\n");
        if ( allocSizes[0] == - 1 )
                CalculateAllocatedSizes(numVariations,sizes,allocSizes);
        for ( i = 0 ; i < numVariations ; i++ )
                fprintf(outputFile,"%d ",allocSizes[i]);
```

```
        fprintf(outputFile,"\n");
        fprintf(outputFile,"%s\n",programName);
        fprintf(outputFile,"%d\n",numSelected);
        for ( i = 0 ; i < numVariations ; i++ )
                fprintf(outputFile,"%d ",numInSites[i]);
        fprintf(outputFile,"\n");
    /*
    ** Now the product bitset.
    */
        bitsetStart = ftell(outputFile);
        fprintf(outputFile,"@PRODUCT_BITSET\n");
        IHBDump(compressed,TestDump,outputFile);
        fprintf(outputFile,"@PROGRAM_INFO\n");
        fprintf(outputFile,"%d\n",bufferSize);
        if ( buffer )
                fwrite(buffer,bufferSize,1,outputFile);
        endOfFile = ftell(outputFile);
        length    = endOfFile - beginingOfFile ;
        fseek(outputFile,beginingOfFile,SEEK_SET);
        fprintf(outputFile,"@BITSET_LENGTH:%010ld\n",length);
    /*
    ** Go back to the begining and write out the header info for the file
    */
        rewind(outputFile);
    /*  fprintf(outputFile,"@NEXT_SLOT:%010ld\n",endOfFile);*/
        fclose(outputFile);
        return 1 ;
    UnableToDumpMaster :
        fprintf(stderr,"WriteOutCompressedBSFile()--Unable to dump master file
    %s\n",masterFileName);
        goto AddTraceback ;
    UnableToOpenFile :
        fprintf(stderr,"WriteOutCompressedBSFile()--Unable to open
```

```
%s\n",outputFileName);
        goto AddTraceback ;
UnableToWriteToFile :
        fprintf(stderr,"WriteOutCompressedBSFile()--Unable to write to output file\n");
        goto AddTraceback ;
UnableToCreateBitSet :
        fprintf(stderr,"WriteOutCompressedBSFile()--Unable to create compressed
bitset\n");
        goto AddTraceback ;
AddTraceback :
        return 0 ;
}

/*
**+E:
**
** Function Name : WriteOutCheckPointFile()
**
** Purpose      : Function will write out a check point(bitset file to
**               the given file.
**
** Usage :        To be used by dbsearch/design programs to write out check
**               points.
**
** Returns      : 1 on success, 0 for failure.
**
** Algorithms   : None.
**
** Revision History :
**
** Author                  Date        Description
** ====================      ========
```

```
    ==================
    ** Fred Soltanshahi          07/25/96      Original version.
    **
    **-E:
 5  */
    int
    WriteOutCheckPointFile(outputFileName,masterFileName,masterRec,programName,bitSet,
    offset,numVariations,sizes,allocSizes,numSelected,numInSites,bufferSize,buffer)
    char *outputFileName ; /* Name of output file */
10  char *masterFileName ; /* Name of master file */
    int  masterRec ;       /* Which record in the master file */
    char *programName ;    /* Name of program generating this. */
    int  *bitSet ;         /* Actual bitset */
    int  offset ;          /* Offset into the bitset */
15  int  numVariations ;   /* Num X01, X02 ... variation sites */
    int  *sizes ;          /* Actuall sizes in each dimension */
    int  *allocSizes ;     /* Allocated sizes(bitset size) in each dimension */
    int  numSelected ;     /* Number of products selected */
    int  *numInSites ;     /* Number of selections in each Y_0? site */
20  int  bufferSize  ;     /* Program specific buffer size */
    int  *buffer    ;      /* Arbitrary data written by the program */
    {
    void   *compressed ;
    /*
25  ** Created a compressed bitset.
    */
          if ( !(compressed = CreateCompressedBitSet(bitSet,
                                                                    offset,


30  numVariations,
                                                                    sizes,


    allocSizes) ) )
```

```
            goto UnableToCreateBitSet ;
        if ( !WriteOutCompressedBSFile(outputFileName,
                                                masterFileName,
                                            masterRec,
5                                               programName,
                                                compressed,
                                                numVariations,
                                                sizes,
                                                allocSizes,
10                                              numSelected,
                                                numInSites,
                                                bufferSize,
                                                buffer))
            goto UnableToWriteToFile ;
15      IHBDestroy(compressed);
        return 1 ;
    UnableToWriteToFile :
        fprintf(stderr,"WriteOutCheckPointFile()--Unable to write to output file\n");
        goto AddTraceback ;
20  UnableToCreateBitSet :
        fprintf(stderr,"WriteOutCheckPointFile()--Unable to create compressed bitset\n");
        goto AddTraceback ;
    AddTraceback :
        return 0 ;
25  }
    static int TestRestore(DumpCode, Parameter, DumpInt)
    int DumpCode;
    void *Parameter;
    int *DumpInt;
30  {
    FILE *fp = (FILE *)Parameter ;
      fscanf(fp, "%d\n", DumpInt);
      return 1 ;
```

```
}
static int GetReagentInfo(char *fileName,char **reagentInfo)
{
FILE *fp ;
char   buffer[4096] ;
int    i = 0 ;
char   *cp ;
       if ( !(fp = fopen(fileName,"r")) )
              return 0 ;
       while ( fgets(buffer,sizeof(buffer)-1,fp))
       {
              buffer[strlen(buffer)-1] = 0 ;
              if ( ( cp = strstr(buffer,"# USER_NAME=")) )
              {
                     (*reagentInfo) = UTL_STR_SAVE(buffer+12);
                     return 1;
              }
/*
** Only read the first 10 lines.
*/
              if ( i++ > 10 )
                     break;
       }
/*
** We did not find the reagent info, lets save an empty string and
** assume the file is old and does not contain the info.
*/
       reagentInfo = UTL_STR_SAVE("");
       return 1 ;
}

/*
**+I:
```

```
**
**
** Function Name : ReadCheckPointFile()
**
** Purpose      : Function will write out a check point(bitset file to
**               the given file.
**
** Usage :       To be used by dbsearch/design programs to write out check
**               points.
**
** Returns      : 1 on success, 0 for failure.
**
** Algorithms   : None.
**
** Revision History :
**
** Author                Date        Description
** ====================      ========
==============
** Fred Soltanshahi      07/25/96    Original version.
**
**-I:
*/
static int
ReadCheckPointFile(inputFileName,inputOffset,masterFileName,masterRec,programName,
bitSet,numVariations,sizes,allocSizes,numSelected,numInSites,masterInfo,bufferSize,progBu
ff)
char *inputFileName ; /* Name of input file */
int  inputOffset ;     /* Offset into the input file where the bitset starts*/
char **masterFileName ; /* Name of master file */
int  *masterRec ;      /* Which record in the master file */
char **programName ;   /* Name of program generating this. */
int  **bitSet ;        /* Actual bitset */
```

```
    int  *numVariations ;   /* Num X01, X02 ... variation sites */
    int  **sizes ;          /* Actuall sizes in each dimension */
    int  **allocSizes ;     /* Allocated sizes(bitset size) in each dimension */
    int  *numSelected ;     /* Number of products selected */
5   int  **numInSites ;     /* Number of selections in each Y_0? site */
    struct MasterFileStruct *masterInfo ;
    int  *bufferSize  ;     /* program specific buffer size */
    int  **progBuff     ;   /* Program specific buffer */
    {
10  FILE   *inputFile ;
    float  version  ;
    long   bitsetSize ;
    time_t startTime ;
    int    i ;
15  int    numBytes ;
    char   buffer[4096] ;
    char   hold[81] ;
    char   *cp ;
           if ( !(inputFile = fopen(inputFileName,"r")))
20                goto UnableToOpenFile ;
    /*
    ** File format is :
    ** @BITSET_START:Where In the File The bitset starts
    ** @VERSION:Version Number:currentVersion Number
25  ** Date/Time Stamp
    ** "Location of the master file"  "Record Number in the file".
    ** @MASTER NumberOfLines  : Number of lines used for the master file.
    ** A copy of master file records(one line at a time,currently 11 lines )
    ** Number of Variation sites
30  ** Actual(current) number of choices for each site.
    ** Allocated number of choices for each site.
    ** Source -- program that generated this file.i.e. dbsearch, dbcsln_des,etc
    ** Number of products selected.
```

```
** Number of selections in each dimension -- num_Y_01_choices num_y_02_choices ...
** @PRODUCT_BITSET
** Bitset -- Row major bitset of products selected.
** @PROGRAM_INFO
** size of program specific buffer
** program specific buffer
*/
/*
** BITSET_START:
*/
        if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                goto UnableToReadFile ;
/*
** VERSION.
*/
        if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                goto UnableToReadFile ;
        cp = strstr(buffer,"@VERSION:");
        if ( !cp )
                goto VersionMissing ;
        version = atof(buffer);
        if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                goto UnableToReadFile ;
        if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                goto UnableToReadFile ;
        sscanf(buffer,"%s %d",hold,masterRec);
        (*masterFileName) = UTL_STR_SAVE(hold);
        if ( masterInfo )
        {
                masterInfo->masterFilePathName = UTL_STR_SAVE(hold);
                masterInfo->masterRecNo  = *masterRec ;
/*
** @MASTER 11  thing.
```

```
    */
                if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                        goto UnableToReadFile ;

    /*
5   ** Reaction class
    */
                if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                        goto UnableToReadFile ;

    /*
10  ** Prefix
    */
                if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                        goto UnableToReadFile ;
                buffer[strlen(buffer)-1] = 0 ;
15              masterInfo->prefixForFiles =
    UTL_STR_SAVE(UTL_FILE_PARSE(buffer,4));
    /*
    ** number of sites
    */
20              if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                        goto UnableToReadFile ;
                masterInfo->numVariationSites = atoi(buffer);

    /*
    ** missing bits
25  */
                if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                        goto UnableToReadFile ;
                masterInfo->numberOfMissingBits = atoi(buffer);

    /*
30  ** Lbits
    */
                if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                        goto UnableToReadFile ;
```

```
                masterInfo->lbits = atoi(buffer);
    /*
    ** core file
    */
                if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                        goto UnableToReadFile ;
                buffer[strlen(buffer)-1] = 0 ;
                masterInfo->corefilePathName = UTL_STR_SAVE(buffer);
    /*
    ** Start core
    */
                if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                        goto UnableToReadFile ;
                masterInfo->startCore = atoi(buffer);
    /*
    ** fingure print file name
    */
                if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                        goto UnableToReadFile ;
                buffer[strlen(buffer)-1] = 0 ;
                masterInfo->fingerFileName = UTL_STR_SAVE(buffer);
    /*
    ** start fingure print
    */
                if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                        goto UnableToReadFile ;
                masterInfo->fingerOffset = atoi(buffer);
                if (!(masterInfo->x_FileName = (char **)UTL_MEM_CALLOC(
                                                masterInfo->numVariationSites,
                                                sizeof(char *))) )
                        goto AddTraceback ;
                if (!(masterInfo->reagentInfo = (char **)UTL_MEM_CALLOC(
                                                masterInfo->numVariationSites,
```

```
                                                    sizeof(char *))) )
                        goto AddTraceback ;
    /*
    ** X1 file pathname
 5  */
                if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                        goto UnableToReadFile ;
                buffer[strlen(buffer)-1] = 0 ;
                masterInfo->x_FileName[0] = UTL_STR_SAVE(buffer);
10              if ( !GetReagentInfo(buffer,&masterInfo->reagentInfo[0]))
                        goto UnableToReadReagentInfo0 ;
    /*
    ** X2 file pathname
    */
15              if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                        goto UnableToReadFile ;
                buffer[strlen(buffer)-1] = 0 ;
                masterInfo->x_FileName[1] = UTL_STR_SAVE(buffer);
                if ( !GetReagentInfo(buffer,&masterInfo->reagentInfo[1]))
20                      goto UnableToReadReagentInfo1;
        }
        else
        {
    /*
25  ** just skip the 12 lines.
    */
                for ( i = 0 ; i < 12 ; i++ )
                {
                        if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
30                              goto UnableToReadFile ;
                }
        }
        if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
```

```
            goto UnableToReadFile ;
        *numVariations = atoi(buffer);
        if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
            goto UnableToReadFile ;
/*
** We have to allocate the arrays for sizes, allocSizes and fragSizes .
*/
        if ( !( (*sizes) = (int *)UTL_MEM_CALLOC(*numVariations,
                                                 sizeof(int))
) )
            goto AddTraceback ;
        if ( !( (*allocSizes) = (int *)UTL_MEM_CALLOC(*numVariations,
                                                 sizeof(int))
) )
            goto AddTraceback ;
        if ( !( (*numInSites) = (int *)UTL_MEM_CALLOC(*numVariations,
                                                 sizeof(int))
) )
            goto AddTraceback ;
    cp = buffer ;
        for ( i = 0 ; i < *numVariations ; i++ )
        {
            sscanf(cp,"%d",&((*sizes)[i]));
            cp = strstr(cp," ");
        }
        if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
            goto UnableToReadFile ;
    cp = buffer ;
        for ( i = 0 ; i < *numVariations ; i++ )
        {
            sscanf(cp,"%d",&((*allocSizes)[i]));
            cp = strstr(cp," ");
        }
```

```
        if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                goto UnableToReadFile ;
                buffer[strlen(buffer)-1] = 0 ;
        (*programName) = UTL_STR_SAVE(buffer);

        if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                goto UnableToReadFile ;
        *numSelected = atoi(buffer);
        if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                goto UnableToReadFile ;
    cp = buffer ;
        for ( i = 0 ; i < *numVariations ; i++ )
        {
                sscanf(cp,"%d",&((*numInSites)[i]));
                cp = strstr(cp," ");
        }
    /*
    ** @PRODUCT_BITSET
    */
        if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                goto UnableToReadFile ;
        IHBRestore(bitSet,TestRestore,inputFile);
    /*
    ** @PROGRAM_INFO
    */
        if ( !fgets(buffer,sizeof(buffer)-1,inputFile))
                goto UnableToReadFile ;
        fclose(inputFile);
        return 1 ;
    VersionMissing :
        fprintf(stderr,"ReadCheckPointFile()--File %s is not a valid ChemSpace
    file\n",inputFileName);
        goto AddTraceback ;
```

```
UnableToReadFile :
        fprintf(stderr,"ReadCheckPointFile()--Unable to read from file\n");
        goto AddTraceback ;
UnableToDumpMaster :
        fprintf(stderr,"ReadCheckPointFile()--Unable to dump master file
%s\n",masterFileName);
        goto AddTraceback ;
UnableToReadReagentInfo0 :
        fprintf(stderr,"ReadCheckPointFile()--Unable to read reagent info in %s\n",
  masterInfo->x_FileName[0]);
UnableToReadReagentInfo1 :
        fprintf(stderr,"ReadCheckPointFile()--Unable to read reagent info in %s\n",
  masterInfo->x_FileName[1]);
        goto AddTraceback ;
UnableToOpenFile :
        fprintf(stderr,"ReadCheckPointFile()--Unable to open %s\n",inputFileName);
        goto AddTraceback ;
UnableToWriteToFile :
        fprintf(stderr,"ReadCheckPointFile()--Unable to write to output file\n");
        goto AddTraceback ;
AddTraceback :
        return 0 ;
}

static struct BitSetFileStruct *ReadAndAllocateMaster(char *masterFileName,int
masterRecNumber,int *initBitset)
{
struct BitSetFileStruct *bitset ;
FILE *mfFP ;
/*
** First allocate everything we need.
*/
        if ( !(bitset = (struct BitSetFileStruct *)UTL_MEM_CALLOC(1,
```

```
                    sizeof(struct BitSetFileStruct ))))
                goto AddTraceback ;
        bitset->masterFileInfo.masterFilePathName = UTL_STR_SAVE(masterFileName);
        bitset->masterFileInfo.masterRecNo = masterRecNumber;
/*
** Some if this info is fixed for now.
*/
        bitset->programInfo.programName = (char *)NULL ;
        bitset->masterFileInfo.numVariationSites = bitset->numVariationSites = 2 ;
        bitset->totalSelected          = 0 ;
    if (!(bitset->masterFileInfo.x_FileName = (char **)UTL_MEM_CALLOC(
                                        bitset->numVariationSites,
                                        sizeof(char *))) )
                goto AddTraceback ;
    if (!(bitset->actuallSizes = (int *)UTL_MEM_CALLOC(
                                        bitset->numVariationSites,
                                        sizeof(int ))) )
                goto AddTraceback ;
    if (!(bitset->allocSizes = (int *)UTL_MEM_CALLOC(
                                        bitset->numVariationSites,
                                        sizeof(int ))) )
                goto AddTraceback ;
    if (!(bitset->numFragsInEachSite = (int *)UTL_MEM_CALLOC(
                                        bitset->numVariationSites,
                                        sizeof(int ))) )
                goto AddTraceback ;
        if ( !RetrieveMasterFile(masterFileName,
                                        mfFP,
                                        masterRecNumber,

&(bitset->masterFileInfo.numberOfMissingBits),
                                        &(bitset->masterFileInfo.lbits),
```

```
    &(bitset->masterFileInfo.corefilePathName),
                                                &(bitset->masterFileInfo.startCore),

    &(bitset->masterFileInfo.fingerFileName),

    &(bitset->masterFileInfo.x_FileName[0]),

    &(bitset->masterFileInfo.x_FileName[1]),
                                                &(bitset->actuallSizes[0]),
                                                &(bitset->actuallSizes[1]),
                                                NULL,
                                                &(bitset->masterFileInfo.fingerOffset),
                                                NULL,
                                                NULL,
                                                NULL,
                                                NULL,
                                                NULL))
            goto AddTraceback ;
        if ( !( bitset->bitset = CreateCompressedBitSet(initBitset,
                                                0,
                                                bitset->numVariationSites,
                                                bitset->actuallSizes,
                                                bitset->allocSizes)) )
            goto AddTraceback ;
        return bitset ;
    AddTraceback :
        fprintf(stderr,"ReadAndAllocateMaster()--Unable to read master file\n");
        return (struct BitSetFileStruct *) NULL ;
    }

    static int CalculateFragsInSties( struct BitSetFileStruct *bitset )
    {
```

```
unsigned char **bitArray ;
int i ;
int j ;
int size ;
int address ;
int *indxs = 0 ;
int what ;
int this ;
        Init();
        if ( !( bitArray = (unsigned char
**)UTL_MEM_CALLOC(bitset->numVariationSites,

sizeof(unsigned char *))) )
                goto AddTraceback ;
        for ( i = 0 ; i < bitset->numVariationSites ; i++ )
        {
                size = ( bitset->actuallSizes[i] + 7 ) / 8 ;
                if ( !(bitArray[i] = (unsigned char
*)UTL_MEM_CALLOC(size,sizeof(unsigned char))))
                        goto AddTraceback ;
        }
        for ( address = -1 , i = 0 ; i < bitset->totalSelected  ; i++ )
        {
                address = IHBFindNextOne(bitset->bitset,address+1);
                BitSetAddressToIndexes(bitset,address,&indxs,0);
/*
** For every hit and every variation site address set the bit to 1.
*/
                for ( j = 0 ; j < bitset->numVariationSites  ; j++ )
                {
                        what = indxs[j] % 8;
                        this = indxs[j] / 8;
                        bitArray[j][this] |= setbits[what];
```

```
            }
        }
        for ( i = 0 ; i < bitset->numVariationSites  ; i++ )
        {
            size = ( bitset->actuallSizes[i] + 7 ) / 8 ;
            bitset->numFragsInEachSite[i] = 0 ;
            for ( j = 0 ; j < size ; j++ )
                bitset->numFragsInEachSite[i] += nbits[bitArray[i][j] & 255];


        }
/*
** Get read of memory we allocated for calculating the hits the last time.
*/
        if ( bitset->fragmentBitset )
        {
            for ( i = 0 ; i < bitset->numVariationSites  ; i++ )
                UTL_MEM_FREE(bitset->fragmentBitset[i]);
            UTL_MEM_FREE(bitset->fragmentBitset );
        }
        bitset->fragmentBitset = bitArray ;
        UTL_MEM_FREE(indxs);
        return 1 ;
AddTraceback :
        return 0 ;
}


static int GetPartialProductsAddresses(struct BitSetFileStruct *bitset,int numFixedSites, int
*fixedSitesIndexes, int site, int **hitIndexes)
{
int i ;
int j ;
```

```
int k ;
int address ;
int *indxs = 0 ;
int skipIt ;
int numHits = 0 ;
unsigned char **bitArray ;
int size ;
int what ;
int this ;
        Init();
        if ( !( bitArray = (unsigned char
**)UTL_MEM_CALLOC(bitset->numVariationSites,

sizeof(unsigned char *))) )
                goto AddTraceback ;
        for ( i = 0 ; i < bitset->numVariationSites ; i++ )
        {
/*
** We only want to count the fragments for the site that is being exploded.
*/
                if ( ( site != -1 ) && ( i != site ) )
                        continue ;
                size = ( bitset->actuallSizes[i] + 7 ) / 8 ;
                if ( !(bitArray[i] = (unsigned char
*)UTL_MEM_CALLOC(size,sizeof(unsigned char))))
                        goto AddTraceback ;
        }
        for ( address = -1 , i = 0 ; i < bitset->totalSelected ; i++ )
        {
                address = IHBFindNextOne(bitset->bitset,address+1);
                BitSetAddressToIndexes(bitset,address,&indxs,0);
/*
** The sites that have already been expanded will constraint what hits
```

```
** we find.
*/
                if ( numFixedSites )
                {
                        skipIt = 0 ;
                        for ( k = 0 ; k < bitset->numVariationSites ; k++ )
                        {
                                if ( fixedSitesIndexes[k] == -1 )
                                        continue ;
/*
** our hit index matches our constraint index.
*/
                                if ( fixedSitesIndexes[k] != indxs[k] )
                                {
                                        skipIt = 1  ;
                                        break;
                                }
                        }
                        if ( skipIt )
                                continue ;
                }
                numHits++ ;
                for ( j = 0 ; j < bitset->numVariationSites  ; j++ )
                {
                        if ( ( site != -1 ) && ( j != site )  )
                                continue ;
                        what = indxs[j] % 8;
                        this = indxs[j] / 8;
                        bitArray[j][this] |= setbits[what];
                }
        }
#if 0
/*
```

```
** Figure out how many hits there are for this site.
*/
        for ( k = 0 ; k < bitset->numVariationSites  ; k++ )
        {
                if (  site != k )
                        continue ;
                size = ( bitset->actuallSizes[k] + 7 ) / 8 ;
                numFragmentsPerSite[k] = 0 ;
                for ( j = 0 ; j < size ; j++ )
                        numFragmentsPerSite[k] += nbits[bitArray[k][j] & 255];
        }
#endif
/*
** Now get the indexes for all the hits.
*/
        if ( !( (*hitIndexes) = (int *)UTL_MEM_CALLOC(numHits,
                                                        sizeof(int)))
)
                goto AddTraceback ;
        numHits = 0 ;
        for ( k = 0 ; k < bitset->numVariationSites  ; k++ )
        {
        if ( site == -1 )
                {
                        if ( fixedSitesIndexes[k] != -1 )
                                continue ;
                }
                else
                {
                        if ( site != k )
                                continue ;
                }
                size = ( bitset->actuallSizes[k] + 7 ) / 8 ;
```

```
                for ( i = 0 ; i < size ; i++ )
                {
                        if ( bitArray[k][i] )
                        {
5   /*
    ** If any bit is set in the byte then we need to figure out what the hits are.
    */
                                for ( j = 0 ; j < 8  ; j++ )
                                {
                                        if ( bitArray[k][i] & setbits[j] )
10                                      {
                                                (*hitIndexes)[numHits++] = i * 8 + j ;
                                        }
                                }
15                      }
                }
        }
        for ( i = 0 ; i < bitset->numVariationSites  ; i++ )
                if ( bitArray[i] )
20                      UTL_MEM_FREE(bitArray[i]);
        UTL_MEM_FREE(bitArray );
        return numHits  ;

    AddTraceback :
        return 0 ;
25  }


    static int GetPartialProductsStats( struct BitSetFileStruct *bitset , int numFixedSites, int
    *fixedSitesIndexes, int *numProducts, int *numFragmentsPerSite)
    {
    int i ;
30  int j ;
```

```
int k ;
int address ;
int *indxs = 0 ;
int skipIt ;
int numHits = 0 ;
unsigned char **bitArray ;
int size ;
int what ;
int this ;
        Init();
        if ( !( bitArray = (unsigned char
**)UTL_MEM_CALLOC(bitset->numVariationSites,

sizeof(unsigned char *))) )
                goto AddTraceback ;
        for ( i = 0 ; i < bitset->numVariationSites ; i++ )
        {
/*
** We only want to count the fragments for the sites that are not being
** exploded.
*/
                if ( fixedSitesIndexes[i] != -1 )
                        continue ;
                size = ( bitset->actuallSizes[i] + 7 ) / 8 ;
                if ( !(bitArray[i] = (unsigned char
*)UTL_MEM_CALLOC(size,sizeof(unsigned char))))
                        goto AddTraceback ;
        }
        for ( address = -1 , i = 0 ; i < bitset->totalSelected  ; i++ )
        {
                address = IHBFindNextOne(bitset->bitset,address+1);
                BitSetAddressToIndexes(bitset,address,&indxs,0);
/*
```

```
** The sites that have already been expanded will constraint what hits
** we find.
*/
            if ( numFixedSites )
            {
                skipIt = 0 ;
                for ( k = 0 ; k < bitset->numVariationSites ; k++ )
                {
                    if ( fixedSitesIndexes[k] == -1 )
                        continue ;
/*
** our hit index matches our constraint index.
*/
                    if ( fixedSitesIndexes[k] != indxs[k] )
                    {
                        skipIt = 1 ;
                        break;
                    }
                }
                if ( skipIt )
                    continue ;
            }
            numHits++ ;
            for ( j = 0 ; j < bitset->numVariationSites ; j++ )
            {
                if ( fixedSitesIndexes[j] != -1 )
                    continue ;
                what = indxs[j] % 8;
                this = indxs[j] / 8;
                bitArray[j][this] |= setbits[what];
            }
        }
        for ( k = 0 ; k < bitset->numVariationSites ; k++ )
```

```
        {
                if ( fixedSitesIndexes[k] != -1 )
                        continue ;
                size = ( bitset->actuallSizes[k] + 7 ) / 8 ;
                numFragmentsPerSite[k] = 0 ;
                for ( j = 0 ; j < size ; j++ )
                        numFragmentsPerSite[k] += nbits[bitArray[k][j] & 255];


        }
        *numProducts = numHits ;
        for ( i = 0 ; i < bitset->numVariationSites  ; i++ )
                if ( bitArray[i] )
                        UTL_MEM_FREE(bitArray[i]);
        UTL_MEM_FREE(bitArray );
        return numHits  ;

AddTraceback :
        return 0 ;
}

static int GetPartialProducts( struct BitSetFileStruct *bitset , int numFixedSites, int
*fixedSitesIndexes, int whichSite, int **siteIndexes)
{
int i ;
int j ;
int k ;
int address ;
int *indxs = 0 ;
int skipIt ;
int numHits = 0 ;
        Init();
        for ( address = -1 , i = 0 ; i < bitset->totalSelected  ; i++ )
```

```
        {
                address = IHBFindNextOne(bitset->bitset,address+1);
                BitSetAddressToIndexes(bitset,address,&indxs,0);
    /*
    ** The sites that have already been expanded will constraint what hits
    ** we find.
    */
                if ( numFixedSites )
                {
                        skipIt = 0 ;
                        for ( k = 0 ; k < bitset->numVariationSites ; k++ )
                        {
                                if ( fixedSitesIndexes[k] == -1 )
                                        continue ;
    /*
    ** our hit index matches our constraint index.
    */
                                if ( fixedSitesIndexes[k] != indxs[k] )
                                {
                                        skipIt = 1  ;
                                        break;
                                }
                        }
                        if ( skipIt )
                                continue ;
                }
                numHits++ ;
        fprintf(stderr,"Got a hit on %d %d %d\n",address,indxs[0],indxs[1]);
        }
        return numHits  ;

    AddTraceback :
        return 0 ;
```

```
}

static GetFragmentsUsedInASite( struct BitSetFileStruct *bitset , int whichSite , int **
indxs)
{
unsigned char *bitArray ;
int i ;
int j ;
int size ;
int *address ;
int numHits = 0 ;
int bit ;
        if (!(address = (int
 *)UTL_MEM_CALLOC(bitset->numFragsInEachSite[whichSite],
                                                                sizeof(int))) )

                goto AddTraceback ;
 /*
 ** Figure out how many ints there are in this bitset.
 */
        size = ( bitset->actuallSizes[whichSite] + 7 ) / 8 ;
        for ( bitArray = bitset->fragmentBitset[whichSite], i = 0 ; i < size ; i++ )
        {
                if ( bitArray[i] )
                {
 /*
 ** If any bit is set in the byte then we need to figure out what the hits are.
 */
                        for ( j = 0 ; j < 8  ; j++ )
                        {
                                if ( bitArray[i] & setbits[j] )
                                {
                                        address[numHits++] = i * 8 + j ;
                                }
```

```
                    }
                }
            }
            (*indxs) = address ;
            return numHits ;
AddTraceback :
            return 0 ;
}


static struct BitSetFileStruct *ReadAndAllocate(char *fileName ,int offset )
{
struct BitSetFileStruct *bitset ;
            if ( !(bitset = (struct BitSetFileStruct *)UTL_MEM_CALLOC(1,
                                sizeof(struct BitSetFileStruct ))))
                goto AddTraceback ;
            if ( !ReadCheckPointFile(fileName,
                                                    offset,

&(bitset->masterFileInfo.masterFilePathName),

&(bitset->masterFileInfo.masterRecNo),
                                                    &(bitset->programInfo.programName),
                                                    &(bitset->bitset),
                                                    &(bitset->numVariationSites),
                                                    &(bitset->actuallSizes),
                                                    &(bitset->allocSizes),
                                                    &(bitset->totalSelected),
                                                    &(bitset->numFragsInEachSite),
                                                    &(bitset->masterFileInfo),
                                                    &(bitset->programInfo.bufferSize),
                                                    NULL))
                goto AddTraceback ;
```

```
        return bitset ;
    AddTraceback :
        return ( struct BitSetFileStruct *)NULL ;
    }

5   static int ReadBitsetCoreInfo(void *bs, char **masterFileName, int *masterRecno, char
    **core, char **xrString, int *numSites, char ***xFileNames )
    {
    struct BitSetFileStruct *bitset = (struct BitSetFileStruct *) bs;
    int    recNo ;
10  FILE   *fp ;
    int    i ;
    int        found = 0 ;
    char   *line ;
    char   *cp ;
15  char   *cp1 ;
        *numSites = bitset->numVariationSites ;
        *masterFileName = bitset->masterFileInfo.masterFilePathName;
        *masterRecno    = bitset->masterFileInfo.masterRecNo;
        if ( !((*xFileNames) = (char **)UTL_MEM_CALLOC(*numSites,


20  sizeof(char *)) ))
                goto AddTraceback ;
            for ( i = 0 ; i < *numSites ; i++ )
                (*xFileNames)[i] =
    UTL_STR_SAVE(bitset->masterFileInfo.x_FileName[i]);
25  /*
    ** Open the core file and read in the core and parse out the XRstring.
    */
        if ( !(fp = fopen(bitset->masterFileInfo.corefilePathName,"r")) )
            goto UnableToReadCore ;
30      recNo = 0 ;
        found = 0 ;
```

```
        while ( -1 != UTL_SCAN_GETS( fp, "\\", "#", &line))
        {
                recNo++ ;
                if ( recNo ==       bitset->masterFileInfo.startCore )
5               {
                        found = 1 ;
                        break;
                }
        }
10      if (!found )
                goto UnableToReadCore ;
    /*
    ** Replace all occurances of Y_0x with Xx.
    */
15      (*core ) = UTL_STR_SAVE(line);
        cp = strstr(line,"XRLIST=");
        if ( !cp )
                (*xrString) = UTL_STR_SAVE("");
        else
20      {
    /*
    ** Skip the first double quote.
    */
                cp += 8 ;
25  /*
    ** Go find the end of double quotes.
    */
                cp1 = cp ;
                while ( (*cp)  != '"')
30                      cp++ ;
                *cp = 0 ;
                (*xrString) = UTL_STR_SAVE(cp1);
         }
```

```
        fclose(fp);
        return 1 ;
    UnableToReadCore :
        fprintf(stderr,"ReadBitsetCoreInfo() -- Unable to read core %s %d\n",
 5      bitset->masterFileInfo.corefilePathName,
        bitset->masterFileInfo.startCore);
    AddTraceback :
        fprintf(stderr,"ReadBitsetCoreInfo() -- Unable to read core info\n");
        return 0 ;
10  }

    static int ReadMasterCoreInfo(char *masterFile, int index, char **core, char **xrString,
    int *numSites, char ***xFileNames )
    {
    int    recNo ;
15  FILE   *fp ;
    int    i ;
    int        found = 0 ;
    char   *line ;
    char   *cp ;
20  char   *cp1 ;
    char *prefix = (char *)NULL ;
    char *coreFile = (char *)NULL ;
    char *fpFileName = (char *)NULL ;
    int  fpOffset ;
25  int  mBits ;
    int  lBits ;
    int  startCore ;
        *numSites = 2 ; /* fixed for now */
        if ( !((*xFileNames) = (char **)UTL_MEM_CALLOC(*numSites,

30  sizeof(char *)) ))
```

```
            goto AddTraceback ;
    /*
    ** Get the master file info.
    */
5       if ( !GetMasterRecordHeader(masterFile,
                                                    index,
                                                    &prefix,
                                                    &mBits,
                                                    &lBits,
10                                                  &coreFile,
                                                    &startCore,
                                                    &(*xFileNames)[0],
                                                    &(*xFileNames)[1],
                                                    numSites,
                                                    &fpFileName,
15                                                  &fpOffset))
            goto AddTraceback ;
    /*
    ** Open the core file and read in the core and parse out the XRstring.
20  */
        if ( !(fp = fopen(coreFile,"r")) )
            goto UnableToReadCore ;
        recNo = 0 ;
        found = 0 ;
25      while ( -1 != UTL_SCAN_GETS( fp, "\\", "#", &line))
        {
            recNo++ ;
            if ( recNo ==       startCore )
            {
30              found = 1 ;
                break;
            }
        }
```

```
        if (!found )
                goto UnableToReadCore ;
        (*core ) = UTL_STR_SAVE(line);
        cp = strstr(line,"XRLIST=");
        if ( !cp )
                (*xrString) = UTL_STR_SAVE("");
        else
        {
    /*
    ** Skip the first double quote.
    */
                cp += 8 ;
    /*
    ** Go find the end of double quotes.
    */
                cp1 = cp ;
                while ( (*cp)  != '"')
                        cp++ ;
                *cp = 0 ;
                (*xrString) = UTL_STR_SAVE(cp1);
          }

          fclose(fp);
        if ( coreFile )
                UTL_MEM_FREE(coreFile);
        if ( fpFileName )
                UTL_MEM_FREE(fpFileName);
        if ( prefix )
                UTL_MEM_FREE(prefix);
          return 1 ;
    UnableToReadCore :
          fprintf(stderr,"ReadMastersetCoreInfo() -- Unable to read core %s %d\n",
                coreFile,startCore);
```

```
AddTraceback :
        fprintf(stderr,"ReadMastersetCoreInfo() -- Unable to read core info\n");
    if ( coreFile )
            UTL_MEM_FREE(coreFile);
    if ( fpFileName )
            UTL_MEM_FREE(fpFileName);
    if ( prefix )
            UTL_MEM_FREE(prefix);
        return 0 ;
}

static void DeallocateBitset( struct BitSetFileStruct *bitset )
{
int i ;
        if ( bitset->masterFileInfo.masterFilePathName )
                UTL_MEM_FREE(bitset->masterFileInfo.masterFilePathName);
        if ( bitset->masterFileInfo.corefilePathName )
                UTL_MEM_FREE(bitset->masterFileInfo.corefilePathName);
        if ( bitset->masterFileInfo.fingerFileName )
                UTL_MEM_FREE(bitset->masterFileInfo.fingerFileName);
        if ( bitset->masterFileInfo.prefixForFiles )
                UTL_MEM_FREE(bitset->masterFileInfo.prefixForFiles);
        for ( i = 0 ; i < bitset->masterFileInfo.numVariationSites ; i++ )
                UTL_MEM_FREE(bitset->masterFileInfo.x_FileName[i]);
        if ( bitset->masterFileInfo.x_FileName)
                UTL_MEM_FREE(bitset->masterFileInfo.x_FileName);
        if ( bitset->programInfo.programName )
                UTL_MEM_FREE(bitset->programInfo.programName);
        if ( bitset->programInfo.buffer )
                UTL_MEM_FREE(bitset->programInfo.buffer);
        IHBDestroy(bitset->bitset);
        if ( bitset->actuallSizes )
                UTL_MEM_FREE(bitset->actuallSizes);
```

```
        if ( bitset->allocSizes )
                UTL_MEM_FREE(bitset->allocSizes);
        if ( bitset->numFragsInEachSite )
                UTL_MEM_FREE(bitset->numFragsInEachSite);
        UTL_MEM_FREE(bitset);
        bitset = (struct BitSetFileStruct *) NULL ;
}
void CS_PRDCT_BITSET_DUMP( struct BitSetFileStruct *bitset )
{
int i ;
int indx ;
int indx1 ;
int indx2 ;
        fprintf(stderr,"Master file name :
%s\n",bitset->masterFileInfo.masterFilePathName);
        fprintf(stderr,"Master file rec  : %d\n",bitset->masterFileInfo.masterRecNo);
        fprintf(stderr,"Program Name     : %s\n",bitset->programInfo.programName);
        fprintf(stderr,"Number of Sites  : %d\n",bitset->numVariationSites);
        fprintf(stderr,"Number Selected  : %d\n",bitset->totalSelected);
        fprintf(stderr,"Actual Sizes     : ");
        for ( i = 0 ; i < bitset->numVariationSites ; i++ )
                fprintf(stderr,"%d ",bitset->actuallSizes[i]);
        fprintf(stderr,"\n");
        fprintf(stderr,"Alloc  Sizes     : ");
        for ( i = 0 ; i < bitset->numVariationSites ; i++ )
                fprintf(stderr,"%d ",bitset->allocSizes[i]);
        fprintf(stderr,"\n");
        fprintf(stderr,"Num Frags in X?  : ");
    /*
    ** If the number of fragments is zero then we will write -1 to tell others
    ** to calculate this themselves.
    */
        for ( i = 0 ; i < bitset->numVariationSites ; i++ )
```

```
                fprintf(stderr,"%d ",(bitset->numFragsInEachSite[i] == 0 )?-1:
                                                        bitset->numFragsInEachSite[i]);
        fprintf(stderr,"\n");
        fprintf(stderr,"Selections      : \n");
        indx = -1 ;
        do
        {
                indx = IHBFindNextOne(bitset->bitset,indx+1);
                if ( indx == -1 )
                        break;
                indx1 = indx / bitset->allocSizes[1] ;
                indx2 = indx % bitset->allocSizes[1] ;
                fprintf(stderr,"%d %d\n",indx1 + 1 ,indx2 + 1 );
        } while ( 1 );
}
void CS_PRDCT_BITSET_GET_HITS( struct BitSetFileStruct *bitset , int **indexes)
{
int i ;
int indx ;
int indx1 ;
int indx2 ;
int hitNo = 0 ;
        indx = -1 ;
        do
        {
                indx = IHBFindNextOne(bitset->bitset,indx+1);
                if ( indx == -1 )
                        break;
                indx1 = indx / bitset->allocSizes[1] ;
                indx2 = indx % bitset->allocSizes[1] ;
                indexes[0][hitNo] = indx1 + 1 ;
                indexes[1][hitNo] = indx2 + 1 ;
                hitNo++ ;
```

```
        } while ( 1 );
    }

    /*
    **+E:
    **
    ** Function Name : CS_PRDCT_BITSET_OPEN()
    **
    ** Purpose       : Function will read in the header for a CS product bitset.
    **
    ** Usage :
    **
    ** Returns       : A handle to the product bitset info structure or NULL on
    **                 error.
    **
    ** Algorithms    : None.
    **
    ** Revision History :
    **
    ** Author                Date        Description
    ** ====================      ========
    ================
    ** Fred Soltanshahi        07/26/96    Original version.
    **
    **-E:
    */
    void *CS_PRDCT_BITSET_OPEN( char *bitsetFileName , int offset )
    {
    struct BitSetFileStruct *bitset ;
        if ( !(bitset = ReadAndAllocate(bitsetFileName,offset)) )
            return (void *)NULL ;
        bitset->totalSelected = IHBCountOnes(bitset->bitset,
```

```
                0, IHBBitSize(bitset->bitset));
    /*
    ** If the program did not keep track of and output this to the file then we
    ** need to calculate it ourselves.
 5  */
        if ( ( bitset->numFragsInEachSite[0] == 0 ) || ( bitset->numFragsInEachSite[0]
    == -1 ) )
        {
            CalculateFragsInSties(bitset);
10      }
        return (void *)bitset ;
    }

    /*
    **+E:
15  **
    **
    ** Function Name : CS_PRDCT_BITSET_CLOSE()
    **
    ** Purpose      : Function will close a bitset file and cleanup allocated.
20  **               memory.
    **
    ** Usage :
    **
    ** Returns      : None.
25  **
    ** Algorithms   : None.
    **
    ** Revision History :
    **
30  ** Author              Date        Description
    ** ==================        ========
    ================
```

```
** Fred Soltanshahi          07/26/96      Original version.
**
**-E:
*/
void CS_PRDCT_BITSET_CLOSE( struct BitSetFileStruct *bitset )
{
        DeallocateBitset(bitset);
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_WRITE()
**
** Purpose       : Function will write a bitset into the given file.
**
** Usage :
**
** Returns       : 1 on success or 0 on failure.
**
** Algorithms    : None.
**
** Revision History :
**
** Author                    Date        Description
** ========================      ========
================
** Fred Soltanshahi          08/02/96      Original version.
**
**-E:
*/
int CS_PRDCT_BITSET_WRITE(char *fileName,char *programName,struct
```

```
BitSetFileStruct *productBitset,int progBufferSize,int *progBuffer)
{
        if ( !WriteOutCompressedBSFile(fileName,

productBitset->masterFileInfo.masterFilePathName,
                                        productBitset->masterFileInfo.masterRecNo,
                                        programName,
                                        productBitset->bitset,
                                        productBitset->numVariationSites,
                                        productBitset->actuallSizes,
                                        productBitset->allocSizes,
                                        productBitset->totalSelected,
                                        productBitset->numFragsInEachSite,
                                        progBufferSize,
                                        progBuffer))
                goto AddTraceback ;
        return 1 ;
AddTraceback :
        fprintf(stderr,"CS_PRDCT_BITSET_WRITE()--Unable to write bitset file\n");
        return 0 ;
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_CREATE()
**
** Purpose      : Function will create an in-memory product bitset from a
**                master file.
**
** Usage :
**
```

```
** Returns      : A handle to the product bitset info structure or NULL on
**               error.
**
** Algorithms   : None.
**
** Revision History :
**
** Author                Date        Description
** ====================      ========
==============
** Fred Soltanshahi      08/02/96    Original version.
**
**-E:
*/
void  *CS_PRDCT_BITSET_CREATE(char *masterFileName,
                              int masterRecNumber,
                              int *initRawBitset)

{
struct BitSetFileStruct *bitset ;
        if ( !(bitset = ReadAndAllocateMaster(masterFileName,

masterRecNumber,
                                              initRawBitset)) )

                return (void *)NULL ;
        else
                return (void *)bitset ;
 }

 /*
 **+E:
 **
 **
 ** Function Name : CS_PRDCT_BITSET_SETBITS()
```

```
**
** Purpose      : Function will copy a raw bitset into the ChemSpace product
**               bitset format.
**
** Usage :
**
** Returns      : 1 on success or zero on failure.
**
** Algorithms   : None.
**
** Revision History :
**
** Author                Date        Description
** ====================      ========
==================
** Fred Soltanshahi      08/02/96    Original version.
**
**-E:
*/
int  CS_PRDCT_BITSET_SETBITS(void *bs, int *rawBS, int numProducts)
{
struct BitSetFileStruct *bitset = (struct BitSetFileStruct *)bs ;
void   *compressed ;
static int    firstTime = 1 ;
int    i ;
int    total;
char   *cp = (char *)rawBS ;
int rowLength ;
int index1 ;
int index2 ;
int byte ;
int bit ;
int totalSelected = 0 ;
```

```
        if ( firstTime )
        {
                Init();
                firstTime = 0 ;
        }
    /*
    ** Just create a new one.
    */
        if ( bitset->bitset )
                IHBDestroy(bitset->bitset ) ;
        if ( !(bitset->bitset  = CreateCompressedBitSet(rawBS,
                                                                0,

    bitset->numVariationSites,

    bitset->actuallSizes,

    bitset->allocSizes) ) )
                goto UnableToCreateBitSet ;
         total = bitset->actuallSizes[0] ;
        for ( i = 1 ; i < bitset->numVariationSites ; i++ )
                total *= bitset->actuallSizes[i] ;
        if ( numProducts == -1 )
    /*
    ** Calculate what products are being set.
    */
        {
                numProducts = 0 ;
                rowLength = bitset->actuallSizes[1] ;
                for ( i = 0 ; i < total ; i++ )
                {

                        byte = ( i ) / 8 ;
```

```
                    bit  = ( i  ) % 8 ;
                    if ( cp[byte] & setbits[bit] )
                        numProducts++ ;
                }
        }
        bitset->totalSelected = numProducts ;
        return 1 ;
UnableToCreateBitSet :
        fprintf(stderr,"CS_PRDCT_BITSET_SETBITS-- Unable to set bit\n");
        return 0 ;
}

/*
**+E:
**
** Function Name : CS_PRDCT_BITSET_TO_RAW ()
**
** Purpose      : Function will copy a ChemSpace product bitset to a
**               raw bitset format.
**
** Usage :     calloc rawBS before call. useAlloc nonzero to use allocated
**            rather than actual dimensions
**
** Returns      : 1 on success or zero on failure.
**
** Algorithms    : None.
**
** Revision History :
**
** Author                  Date        Description
** ====================      ========
==============
```

```
** David Patterson          09/09/96      Original version.
**
**-E:
*/
int  CS_PRDCT_BITSET_TO_RAW (void *bs, int *rawBS, int useAlloc)
{
 CS_PRDCT_BITSET_CONCAT_RAW(bs, rawBS, 0, useAlloc);
 return 1;
}

int CS_PRDCT_BITSET_CONCAT_RAW(void *bs, int *rawBS, int offset,
                               int useAlloc)
{
 int *indxs = 0;
 int address, sum, b;
 struct BitSetFileStruct *bitset = (struct BitSetFileStruct *) bs;
  for ( address = -1 , b = 0 ; b < bitset->totalSelected  ; b++ )
  {
            address = IHBFindNextOne(bitset->bitset,address+1);
            BitSetAddressToIndexes(bitset,address,&indxs,0);
            if (useAlloc)
              FlagProduct(rawBS, 0,0, address+offset);
            else  /* must explicitly calculate the address */
              {sum= CS_PRDCT_BITSET_INDEXES_TO_INDEX( bitset, indxs) ;
               FlagProduct(rawBS, 0,0, sum+offset);
              }
  }
  UTL_MEM_FREE(indxs);
  return 1;
}

/*
**+E:
```

```
**
**
** Function Name : CS_PRDCT_BITSET_SELECTED ()
**
** Purpose       : Function will return a ChemSpace bitset's totalSelected
**
** Usage :
**
** Returns       : integet count of selected bits in bitset
**
** Algorithms    : None.
**
** Revision History :
**
** Author                Date        Description
** ====================        ========
==============
** David Patterson       09/24/96    Original version.
**
**-E:
*/
int  CS_PRDCT_BITSET_SELECTED (void *bsvoid )
{
struct BitSetFileStruct  *bs = (struct BitSetFileStruct *) bsvoid;
        return bs->totalSelected;
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_REVEAL ()
**
```

```
** Purpose      : Function will return a ChemSpace bitset's struct info to
**               external calling program.
**
** Usage :
**
** Returns      : 1 on success or zero on failure.
**
** Algorithms   : None.
**
** Revision History :
**
** Author                Date        Description
** ====================      ========
==============
** David Patterson       09/10/96    Original version.
**
**-E:
*/
int  CS_PRDCT_BITSET_REVEAL (void *bsvoid,
        char **MasterFile_Bitset,
        int  *StartRec_Bitset,
         int *BitsInAbsentia,
         int *BitsInAbsentiaNoCount,
         char **CoreFile,
         int  *StartCore,
         char **FngrFile,
         char ***Xfiles,
         int **nY,
         FILE **FngrFile_File,
         int  *FingerOff,
         char **ScreenFileName,
         int  *BytesPerFingerPrint,
         int  *WordsPerFingerprint,
```

```
        int **query,
        int **FingerCore_FP,
        int *FingerCore_Card    )
{
int i, size;
int *fooi;
struct BitSetFileStruct  *bs = (struct BitSetFileStruct *) bsvoid;
if (MasterFile_Bitset)
  *MasterFile_Bitset   = bs->masterFileInfo.masterFilePathName ;
if (StartRec_Bitset)
  *StartRec_Bitset     =bs->masterFileInfo.masterRecNo ;
if (BitsInAbsentia)
  *BitsInAbsentia       = bs->masterFileInfo.numberOfMissingBits;
if (BitsInAbsentiaNoCount)
  *BitsInAbsentiaNoCount = bs->masterFileInfo.lbits;
if (CoreFile)
  *CoreFile             = bs->masterFileInfo.corefilePathName;
if (StartCore)
  *StartCore            = bs->masterFileInfo.startCore;
if (FngrFile)
  *FngrFile             = bs->masterFileInfo.fingerFileName;
if (Xfiles)
  *Xfiles               = bs->masterFileInfo.x_FileName;
if (nY)
  *nY                   = bs->actuallSizes;
if ( FngrFile_File)
  {     if (!((*FngrFile_File) = UTL_FILE_FOPEN((*FngrFile),"r"))) return 0;
        if (!UTL_FILE_FREAD(&i,sizeof(int),1,*FngrFile_File)) return 0; /* nbits in
fp */
        *BytesPerFingerPrint = ( i + 7 ) / 8 ;
        *WordsPerFingerprint = (*BytesPerFingerPrint + 3) / 4;
        (*query) = (int     *) UTL_MEM_ALLOC( *BytesPerFingerPrint);
```

```
            if (!UTL_FILE_FREAD(&i,sizeof(int),1,*FngrFile_File)) return 0; /* record
    cnt  */
            if (!UTL_FILE_FREAD(&i,sizeof(int),1,*FngrFile_File)) return 0; /* record
    size */
            rewind(*FngrFile_File);
            if (!(fooi = (int *) UTL_MEM_ALLOC( i              ))) return 0;
            size = (3+i)/4 ;
            for ( i=0; i<= *FingerOff; i++)
                  if (!UTL_FILE_FREAD( fooi,sizeof(int),size,*FngrFile_File))
                        return 0;
    /*  if ( fooi[1] != 2 + nY_01 * nY_02 ) return 0; */
            if ( ScreenFileName )
            {
                  if (!((*ScreenFileName) = UTL_STR_SAVE(fooi+4))) return 0 ;
            }

            if ( FingerCore_FP )
            {
                  *FingerCore_FP = fooi;
                  if (!UTL_FILE_FREAD( FingerCore_Card,sizeof(int),1,
    *FngrFile_File))
                        return 0;
                  if (!UTL_FILE_FREAD(*FingerCore_FP  ,
                                          sizeof(int),
                                          *WordsPerFingerprint,
                        *FngrFile_File))
                        return 0;
            }
      }
          return 1;
    }

    /*
```

```
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_INDEXES_TO_INDEX()
**
** Purpose      : Function will return the right bit given a set of indices
**
** Usage :      all indexes are 0 based.
**
** Returns      : index to use in bitset.
**
** Algorithms   : None.
**
** Revision History : extracted from CS_PRDCT_BITSET_SET_PRD_BIT by
**                    David Patterson
**
** Author                  Date        Description
** ======================      =========
================
** Fred Soltanshahi        08/02/96    Original version.
**
**-E:
*/
int  CS_PRDCT_BITSET_INDEXES_TO_INDEX( struct BitSetFileStruct *bitset,
                                       int *indexes)
{
int   i ;
int   j ;
int   rowLength[MAX_VARIATION_SITES] ;
int   indx = 0 ;
      for ( i = 0 ;  i < bitset->numVariationSites ;  i++  )
      {
            rowLength[i] = 1 ;
```

```
                for ( j = i + 1 ; j < bitset->numVariationSites  ; j++ )
                        rowLength[i] *= bitset->actuallSizes[j] ;
        }
        for ( i = 0 ;  i < bitset->numVariationSites ;  i++ )
        {
                indx += indexes[i] * rowLength[i] ;
        }
        return indx ;
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_ALLOC_SIZE_INDEXES_TO_INDEX()
**
** Purpose      : Function will return the right bit given a set of indices
**              it uses the allocated sizes in the bitset to get the info.
**
*/
int  CS_PRDCT_BITSET_ALLOC_SIZE_INDEXES_TO_INDEX( struct BitSetFileStruct
*bitset,
                                        int *indexes)
{
int   i ;
int   j ;
int   rowLength[MAX_VARIATION_SITES] ;
int   indx = 0 ;
        for ( i = 0 ;  i < bitset->numVariationSites ;  i++  )
        {
                rowLength[i] = 1 ;
                for ( j = i + 1 ; j < bitset->numVariationSites  ; j++ )
                        rowLength[i] *= bitset->allocSizes[j] ;
```

```
        }
        for ( i = 0 ;  i < bitset->numVariationSites ;  i++ )
        {
                indx += indexes[i] * rowLength[i] ;
        }
        return indx ;
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_SET_PRD_BIT()
**
** Purpose      : Function will set a product bit with the  given indexes.
**
** Usage :
**
** Returns      : none.
**
** Algorithms   : None.
**
** Revision History :
**
** Author                Date        Description
** ====================          ========
==============
** Fred Soltanshahi        08/02/96      Original version.
**
**-E:
*/
int  CS_PRDCT_BITSET_SET_PRD_BIT(void *bs, int *indexes)
{
```

```
    struct BitSetFileStruct *bitset = (struct BitSetFileStruct *)bs ;
    int    indx = 0 ;
           indx = CS_PRDCT_BITSET_ALLOC_SIZE_INDEXES_TO_INDEX(bitset,
    indexes);
           IHBSet(bitset->bitset, indx );
           bitset->totalSelected++ ;
           return 1 ;
    }

    /*
    **+E:
    **
    **
    ** Function Name : CS_PRDCT_BITSET_GET_RINFO()
    **
    ** Purpose      : Function will return the Reaction/Reagent info from
    **               the bitset file.
    **
    ** Usage :
    **
    ** Returns      : none.
    **
    ** Algorithms   : None.
    **
    ** Revision History :
    **
    ** Author                  Date        Description
    ** ====================    ========
    ================
    ** Fred Soltanshahi        01/03/97    Original version.
    **
    **-E:
    */
```

```
int  CS_PRDCT_BITSET_GET_RINFO(void *bs, char **reactionInfo,char ***reagentInfo)
{
struct BitSetFileStruct *bitset = (struct BitSetFileStruct *)bs ;
        *reactionInfo = bitset->masterFileInfo.prefixForFiles ;
        *reagentInfo  =  bitset->masterFileInfo.reagentInfo ;
        return 1 ;
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_GET_STATS()
**
** Purpose      : Function will return the statistics for a bitset file,
**               these will include numberOfSites, originalSizes,
**               numberOfProducts and Number of fragments used at each
**               variation site.
**
** Usage :
**
** Returns      : none.
**
** Algorithms   : None.
**
** Revision History :
**
** Author                Date        Description
** ====================        ========
================
** Fred Soltanshahi      08/05/96    Original version.
**
**-E:
```

```
*/
int  CS_PRDCT_BITSET_GET_STATS(void *bs, int *numSites, int *numProducts,
int **sizes, int **numUsed )
{
5   struct BitSetFileStruct *bitset = (struct BitSetFileStruct *)bs ;
        *numSites = bitset->numVariationSites ;
        *numProducts = bitset->totalSelected ;
    /*
    ** Allocate buffers, if they have not been.
10  */
        if ( !(*sizes) )
        {
            if ( !((*sizes) = (int *)UTL_MEM_CALLOC(*numSites,sizeof(int))))
                goto UnableToAllocateMemory ;
15      }
        if ( !(*numUsed) )
        {
            if ( !((*numUsed) = (int *)UTL_MEM_CALLOC(*numSites,sizeof(int))))
                goto UnableToAllocateMemory ;
20      }
        memcpy(*sizes, bitset->actuallSizes, sizeof(int) * *numSites );
        memcpy(*numUsed, bitset->numFragsInEachSite, sizeof(int) * *numSites );
        return 1 ;
    UnableToAllocateMemory :
25      fprintf(stderr,"CS_PRDCT_BITSET_GET_STATS() -- Unable to allocate
    memory\n");
        return 0 ;
    }


    /*
30  **+E:
    **
```

```
**
** Function Name : CS_PRDCT_BITSET_CORE_INFO()
**
** Purpose      : Function will get the xfile and core and xrstring info
**                from the bitset file.
**
** Usage :
**
** Returns      : 1 on success or 0 on error.
**
** Algorithms   : None.
**
** Revision History :
**
** Author                Date        Description
** ====================      ========
==============
** Fred Soltanshahi      08/09/96    Original version.
**
**-E:
*/
int CS_PRDCT_BITSET_CORE_INFO(void *bs, char **masterName, int *masterRecno,
char **core, char **xrString, int *numSites, char ***xFileNames )
{
        return ( ReadBitsetCoreInfo(bs,masterName,masterRecno,
                        core,xrString,numSites,xFileNames));
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_PROG_NAME()
```

```
**
** Purpose      : Function will get the program name that produced this bitset.
**
** Usage :
**
** Returns      : 1 on success or 0 on error.
**
** Algorithms   : None.
**
** Revision History :
**
** Author                Date        Description
** ====================      =========
================
** Fred Soltanshahi        08/09/96      Original version.
**
**-E:
*/
int CS_PRDCT_BITSET_PROG_NAME(void *bs, char **programName)
{
        *programName =  ((struct BitSetFileStruct *)bs)->programInfo.programName ;
        return 1 ;
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_MSTR_CORE_INFO()
**
** Purpose      : Function will get the xfile and core and xrstring info
**               from the master file.
**
```

```
** Usage :
**
** Returns      : 1 on success or 0 on error.
**
** Algorithms   : None.
**
** Revision History :
**
** Author                Date        Description
** ====================      ========
==========================
** Fred Soltanshahi        08/09/96    Original version.
**
**-E:
*/
int CS_PRDCT_MSTR_CORE_INFO(char *masterFile, int index, char **core, char
**xrString, int *numSites, char ***xFileNames )
{
        return ( ReadMasterCoreInfo(masterFile,index
,core,xrString,numSites,xFileNames));
}



/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_CREATE_BIT_STRING()
**
** Purpose      : Function will create a compressed version of a raw bit set.
**               It returns the memory size needed to hold the bitset.
**
** Usage :
```

```
**
** Returns      : pointer to a compressed bitset(this is not a ChemSpace
**               product bitset but just a compressed bitstring)
**
** Algorithms   : None.
**
** Revision History :
**
** Author                Date        Description
** ====================      ========
==============
** Fred Soltanshahi      08/06/96    Original version.
**
**-E:
*/
void  *CS_PRDCT_BITSET_CREATE_BIT_STRING( int *rawBits, int offset, int
numVariations, int *sizes, int *allocSizes, int *totalSize)
{
void *compressed ;
        if ( !(compressed = CreateCompressedBitSet(rawBits,
                                                                offset,

numVariations,
                                                                sizes,

allocSizes) ) )
                goto UnableToCreateBitSet ;
        *totalSize = IHBRealSize(compressed);
        return compressed ;
UnableToCreateBitSet :
        fprintf(stderr,"CS_PRDCT_BITSET_CREATE_BIT_STRING() -- Unable to create
bitset\n");
        return ( void *)NULL ;
```

```
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_DESTROY_BIT_STRING()
**
** Purpose      : Function will destroy the memory for a bitstring
**               allocate by the CREATE call above.
**
** Usage :
**
** Returns      : none
**
** Algorithms   : None.
**
** Revision History :
**
** Author                Date        Description
** ====================      ========
==============
** Fred Soltanshahi        08/06/96     Original version.
**
**-E:
*/
void  CS_PRDCT_BITSET_DESTROY_BIT_STRING( void *bitset)
{
        IHBDestroy(bitset);
}

/*
**+E:
```

```
**
**
** Function Name : CS_PRDCT_BITSET_GETHITS()
**
** Purpose      : Function will return the indexes(into the original X1,X2 files
**               for the requested number of hits.
**
** Usage :
**
** Returns      : Number of hits found or -1 for error.
**
** Algorithms   : None.
**
** Revision History :
**
** Author                Date        Description
** ====================      ========
================
** Fred Soltanshahi      08/07/96    Original version.
**
**-E:
*/
int  CS_PRDCT_BITSET_GETHITS( void *bs, int offset, int numberOfHits, int
***hitIndexes)
{
struct BitSetFileStruct *bitset = (struct BitSetFileStruct *)bs ;
int numFound ;
int numConnections ;
static int *bitAddresses = (int *)NULL ;
static int numBitAddresses = 0 ;
int *indxs = (int *)NULL ;
int start ;
int count ;
```

```
    int i ;
    int j ;
        (*hitIndexes) = (int **)NULL ;
        numConnections = bitset->numVariationSites ;
5   /*
    ** Local housekeeping .
    */
        if ( numberOfHits > numBitAddresses )
        {
10          if ( !bitAddresses )
            {
                if ( !(bitAddresses = (int *)UTL_MEM_CALLOC(numberOfHits,

    sizeof(int))) )
                    goto UnableToAllocate ;

15          }
            else
            {
                if ( !(bitAddresses = (int *)UTL_MEM_REALLOC(bitAddresses,

20  numberOfHits* sizeof(int))) )
                    goto UnableToAllocate ;
            }
            numBitAddresses = numberOfHits ;
        }
25  /*
    ** Figure out if we have the number of hits he wanted and what their addresses
    ** are in the bitset file.
    **
    ** We will have to come back and speed this up if it is to slow, but for now
30  ** lets get it working.
    **
```

```
    */
    /*    start = bitset->firstHitAddress  ; */
          start = -1 ; /* start from the begining */
          for ( count = 0 ; count < offset ; count++ )
          {
                start = IHBFindNextOne(bitset->bitset,start+1);
    /*
    ** Lets remember where the first hit is, this should save us some time later.
    */
                if ( bitset->firstHitAddress <= 0 )
                      bitset->firstHitAddress = start ;
                if ( start == -1 )
                {
                      return 0 ;
                }
          }
    /*
    ** Now lets see how many bits are set from here on.
    */
          for ( numFound = 0 ; numFound < numberOfHits ; numFound++ )
          {
                start= IHBFindNextOne(bitset->bitset,start+1);
                if ( start == -1 )
                      break;
                bitAddresses[numFound] = start ;
          }
    /*
    ** Allocate the arrays.
    */
          if ( !(*hitIndexes = (int **)UTL_MEM_CALLOC(numConnections,sizeof(int *))) )
                goto UnableToAllocate ;
          for ( i = 0 ; i < numConnections ; i++ )
          {
```

```
            if ( !((*hitIndexes)[i] = (int *)UTL_MEM_CALLOC(numFound,

    sizeof(int ))) )
                        goto UnableToAllocate ;
5       }
    /*
    ** Now translate each one of the bitset addresses to the variation site
    ** indexes.
    */
10      for ( i = 0 ; i < numFound ; i++ )
        {
            BitSetAddressToIndexes(bitset,bitAddresses[i],&indxs,0);
            for ( j = 0 ; j < numConnections ; j++ )
                (*hitIndexes)[j][i] = indxs[j] + 1 ; /* Translate to 1 based indexes */
15      }
        if ( indxs )
            UTL_MEM_FREE( indxs );
        return numFound ;
    UnableToAllocate :
20  AddTraceback :
        if ( indxs )
            UTL_MEM_FREE( indxs );
        return -1 ;
    }

25  /*
    **+E:
    **
    **
    ** Function Name : CS_PRDCT_BITSET_GET_PARTIAL_HITS()
30  **
    ** Purpose      : Function will return the indexes(into the original X1,X2 files
    **               for the requested number of hits.
```

```
**
** Usage :
**
** Returns      : Number of hits found or -1 for error.
**
** Algorithms   : None.
**
** Revision History :
**
** Author                Date        Description
** ====================      ========
==============
** Fred Soltanshahi      08/07/96    Original version.
**
**-E:
*/
int  CS_PRDCT_BITSET_GET_PARTIAL_HITS( void *bs, int *numProducts, int site, int
numFixedSites, int *fixedSitesIndexes, int *numFragmentsPerSite, int **hitIndexes )
{
struct BitSetFileStruct *bitset = (struct BitSetFileStruct *)bs ;
int    total ;
       (*hitIndexes) = (int *)NULL ;
       GetPartialProductsStats( bitset ,
                                                    numFixedSites,
                                                    fixedSitesIndexes,
                                                    &total,
                                                    numFragmentsPerSite);

       (*numProducts ) = GetPartialProductsAddresses(bitset,
                                                    numFixedSites,
                                                    fixedSitesIndexes,
                                                    site,
                                                    hitIndexes);
```

```
        return 1 ;
    }

    /*
    **+E:
    **
    ** Function Name : CS_PRDCT_BITSET_GET_PRDCT_PARTIAL_HITS()
    **
    ** Purpose      : Function will return the indexes(into the original X1,X2 files
    **                for the requested number of hits.
    **
    ** Usage :        This works when the csln is actually being exploded.
    **
    ** Returns      : Number of hits found or -1 for error.
    **
    ** Algorithms   : None.
    **
    ** Revision History :
    **
    ** Author                Date        Description
    ** ====================      ========
    ================
    ** Fred Soltanshahi        08/07/96    Original version.
    **
    **-E:
    */
    int  CS_PRDCT_BITSET_GET_PRDCT_PARTIAL_HITS( void *bs, int *numProducts, int
    site, int numFixedSites, int *fixedSitesIndexes, int *numFragmentsPerSite, int **hitIndexes
    )
    {
    struct BitSetFileStruct *bitset = (struct BitSetFileStruct *)bs ;
    int    total ;
```

```
        (*hitIndexes) = (int *)NULL ;
        GetPartialProductsStats( bitset ,
                                                numFixedSites,
                                                fixedSitesIndexes,
                                                &total,
5                                               numFragmentsPerSite);

        (*numProducts ) = GetPartialProductsAddresses(bitset,
                                                numFixedSites,
                                                fixedSitesIndexes,
                                                site,
10                                              hitIndexes);

        (*numProducts ) = total ;
            return 1 ;
    }

15  /* +E
    Abstract: For Chemspace bitset file call callback with products choices not selected.
    Input:
    1. This function takes a BitSetFileStruct returned most likely from:
                CS_PRDCT_BITSET_OPEN(char *filename)
20  2. A void pointer which is passed to callback function.  This is for
        whatever you want.
    3. A pointer to function returning:
                        int (void *udata, int numVariants, int *choices ).
        choices is of size numVariants, the choices are zero based, and
25      choices[0] is the choice for markush Y_01, choice[1] for Y_02 etc.
        NOTE 1: numVariants of -1 and a null for choices is passed to signify
            the end of the choices excluded, just in case the function
            want to do some special processing at the end.
        NOTE 2: The return value from the callback function is ignored.
30  Returns:
        Total number of bits excluded.
```

```
        -1 upon error.
    ** Author                Date        Description
    ** ====================      ========
    ================
5   ** Rob Jilek             07/26/96    Original version.
    */
    int CS_PRDCT_BITSET_ZERO(struct BitSetFileStruct *bitset, void *udata,
    int (*ZeroProducts)(void *udata, int numVariants, int *choices ) )
    {
10          BIT_TRACKING bt[1];
            if ( bitset->numVariationSites <= 0 )
                    return -1;
            bt->numVariations = bitset->numVariationSites;
            bt->bitset = bitset;
15          bt->call_udata = udata;
            bt->funcptr = ZeroProducts;
            bt->choices = (int *) UTL_MEM_CALLOC(bt->numVariations, sizeof(int) );
            bt->totalExcluded = 0;
                    /* The sequence is as follows:
20                          IHBRange has a loop to find zeros/ones.
                                    It calls RangeCallback
                                            RangeCallback calls ZeroProducts callback.
                            while ( not end of list )
                                    call RangeCallback with start and end Range.
                                            for ( i = startRange; i <= EndRange; i++ ) //
25
        RangeCallback
                                                    calculate product array.
                                                    call ZeroProducts                    //

        ZeroCallback
30                  */

                IHBRange(bitset->bitset, 0, (void *) bt, RangeCallback );
                UTL_MEM_FREE((char *) bt->choices );
```

```
        return bt->totalExcluded;
    }
    /*+I
    Synopsis:  Gets called for each range of bits set.  It then
    converts each bit to a product array and calls callback for each.
    */
    static int RangeCallback ( void *udata, int startRange, int endRange )
    {
        BIT_TRACKING *bt = (BIT_TRACKING *) udata;
        int indx;
        int oor;
        int skip;
        void *call_udata;
        int numVar;
        int *choices;
        void *bitset;
        call_udata = bt->call_udata;
        numVar = bt->numVariations;
        choices = bt->choices;
        bitset = bt->bitset;
        for ( indx = startRange; indx <= endRange;  )
        {
            skip = BitSetAddressToIndexes(bitset,indx,&choices,&oor);
            if ( !oor )
            {
                (*bt->funcptr)(call_udata, numVar, choices );
                bt->totalExcluded++;
                indx++;
            }
            else
            {
                if ( skip > 0 )
                    indx += skip;
```

```
                else
                        indx++;
                }
        }
        (*bt->funcptr)(call_udata,-1, (int *) 0 );          /* Signify end of zeros. */
        return 0;
}

/* +E
Abstract: For Chemspace bitset file call callback with products choices selected.
Input:
1. This function takes a BitSetFileStruct returned most likely from:
                CS_PRDCT_BITSET_OPEN(char *filename)
2. A void pointer which is passed to callback function.  This is for
        whatever you want.
3. A pointer to function returning:
                        int (void *udata, int numVariants, int *choices ).
        choices is of size numVariants, the choices are zero based, and
        choices[0] is the choice for markush Y_01, choice[1] for Y_02 etc.
        NOTE 1: numVariants of -1 and a null for choices is passed to signify
                the end of the choices excluded, just in case the function
                want to do some special processing at the end.
        NOTE 2: The return value from the callback function is ignored.
Returns:
        Total number of bits included.
        -1 upon error.
See Also: CS_PRDCT_BITSET_ZERO
** Author                Date        Description
** ====================        ========
==========
** Rob Jilek             08/19/96    Original version.
*/
int CS_PRDCT_BITSET_ONE(struct BitSetFileStruct *bitset, void *udata,
```

```
int (*OneProducts)(void *udata, int numVariants, int *choices ) )
{
        BIT_TRACKING bt[1];
        if ( bitset->numVariationSites <= 0 )
                return -1;
        bt->numVariations = bitset->numVariationSites;
        bt->bitset = bitset;
        bt->call_udata = udata;
        bt->funcptr = OneProducts;
        bt->choices = (int *) UTL_MEM_CALLOC(bt->numVariations, sizeof(int) );
        bt->totalExcluded = 0;
        IHBRange(bitset->bitset, 1, (void *) bt, RangeCallback );
        UTL_MEM_FREE((char *) bt->choices );
        return bt->totalExcluded;
}

#if 0
main(argc,argv)
int argc ;
char *argv[] ;
{
void *h ;
char *masterFileName =
"/home7/fred/work/ADS/dserv/source/dbcsln_des/TestData/Di_300_400.mf" ;
int   masterRecNumber = 1 ;
int  *bitset ;
int  size = ( 300 * 400 + 7 ) / 8 ;
int  i ;
int  j ;
int  indexes[2] ;
char hold[81];
#if 1
        if ( !(h = CS_PRDCT_BITSET_OPEN(argv[1],0)))
```

```
        {
                fprintf(stderr,"Unable to open the bitset file %s\n",argv[1]);
                exit ;
        }
        CS_PRDCT_BITSET_DUMP(h);
        CS_PRDCT_BITSET_CLOSE(h);
#else
        if ( !( h =
CS_PRDCT_BITSET_CREATE(masterFileName,masterRecNumber,NULL) ) )
        {
                fprintf(stderr,"Unable to create bitset for %s\n",masterFileName);
                exit ;
        }
        CS_PRDCT_BITSET_WRITE("Test.bs","MyProg",h,0,NULL);

        indexes[0] = 59 ;
        indexes[1] = 129 ;
      CS_PRDCT_BITSET_SET_PRD_BIT(h,indexes);
        indexes[0] = 159 ;
        indexes[1] = 241 ;
     CS_PRDCT_BITSET_SET_PRD_BIT(h,indexes);
        CS_PRDCT_BITSET_WRITE("Test2.bs","MyProg",h,0,NULL);
        bitset = (int *)UTL_MEM_CALLOC(size,sizeof(int));
        bitset[5] = 49 ;
        bitset[1] = 99 ;
     CS_PRDCT_BITSET_SETBITS(h,bitset,-1);
        CS_PRDCT_BITSET_WRITE("Test1.bs","MyProg",h,0,NULL);
        CS_PRDCT_BITSET_CLOSE(h);
#endif
}
#endif
```

Appendix "S"

```
/*****************************************************************************
*/
/*                          topsim                                           */
/*****************************************************************************
*/
/*+C
 *
 * This program determines which csln "products" are similar to an input
 * structure, where similarity occurs if the sum of differences in encoded
 * "CoMFA" fields is less than some threshold.
 *
 * The csln components are referenced in a master file with
 * one multiline record per cSLN. Record format is
 *  Reaction class xxxx              (where "Reaction class" is a literal)
 *  reaction_name
 *  number_of_sv_sites
 *  missing_bits_count
 *  hashed_only_missing_bits_count
 *  core_filename
 *  core_filename_index_of_core
 *  fingerprint_filename
 *  offset_into_fingerprint_file
 *  first_sv_file_X1
 *  secod_sv_file_X2                 (etc if more than two sv_sites)
 *
 * NOTE -- ALL subsequent entries in the master file whose Reaction class
 * matches the Reaction class of the record referenced by -index are also
 * processed! ("Matching" implies matching of possible other input symbols
 * to components of the Reaction class line.)
 *
 * The input structure is read as encoded fields from stdin (or
 * a named file if provided), one field per line. There
```

```
* must be provided (by a SYBYL SPL script), in order:
*
* "number_of_sv_sites" * "number_of_field_types" fields describing the "core" of the
query
* "number_of_sv_sites" - 1 sextets of relative coordinates of core attachment atoms
* "number_of_sv_sites" * "number_of_field_types" fields describing the "side chains"
*
* Options:
*
*     -master name        - name is the file with master file records
*     -bitset name        - name is a result of an earlier search operation
*                         (use EITHER master or bitset)
*
*     -index number       - which sequential record in master file to begin at
*                         OR offset into bitset in a bitset file
*                         (default = 1)
*
*     -reaction name      - records in master file to be processed must have this
*                         class name
*
*     -details  name      - if provided, records in master file to be processed
*                           must have any one of these tokens following its class name
*
*     -distance tan       - tan is the overall similarity threshold
*                         (default is 90.0)
*
*     -cooweight cwt      - weight of the core attachment coordinates,
*                           relative to fields
*
*     -nocore nocore            - do not consider core topomer differences
*                           By default these are considered (required)
*
*     -allcores allc      - process all cores in the core file
```

```
     *                          By default only one core (index in the master file) is processed
     *
     *     -maxhits max         - stop when max hits are found (default infinity)
     *
     *     -input filename      - name of file with queries (default stdin)
     *
     *     -output filename     - specifies the output file for the hit info
     *
     *      -#                  This flag forces the display of all
     *                          options
     *
     *
     ******************************************************************************
     /
     #include <stdio.h>
     #include <signal.h>
     #include <ctype.h>
     #include <unistd.h>
     #include <string.h>
     #include <sys/stat.h>
     #include <math.h>
     #include "parseopt.h"
     #include "utl_str.h"
     #include "utl_mem.h"
     #include "utl_file.h"
     #include "utl_math.h"
     #include "ct.h"
     #include "ct_expr.h"
     #include "ct_proto.h"
     #define GoodExit 0
     #define ErrorExit 1
     #define Visual(s) {                fprintf s; }
```

```
static FILE     *OutputFile = 0;
static char     *OutputFileBase;
static char     OutputFileName[200];
static int       nOutFiles = 0;  /* number of output files */
static char     *MasterFile = 0;
static char     *BitsetFile = 0;
void            *bitset;
static int      MasterRecord = 1;
static FILE      *MasterFile_File;
static int      StartCore;
static char     *InputSource = 0;
static FILE      *InputSourceFile;
static char      *ReactionNeeded = 0;
static char      *ScratchDetails = 0;
static int       nDetail = 0;
static char      **ReactionDetails = 0;
static char      *XWeights = 0;
static double    *RWeights = 0;
static double    CoreWeight;
static char      *FieldTypes = 0;
static int       nFType = 0;
static char      **FTypes = 0;
static double    *FWeights = 0;
static char      **FOrder = 0; /* temp, for recording L->R order of data
                                  in side chain SLN */
static char      **FROrder = 0;
static char     *Corefile = 0;
static FILE      *CoreFile_File;
static char     *CoreNow;
static char      **Xfile;
static char      **Xname;
static double    Distance = 90.0;
static double    CoreDistance = 0.0;
```

```
static double   DWeight = 1.0;
static double   Dist[16][16];
static double   boundary[16];
static double   CXcoords[6], CXdiffsq[6];
static double   searched = 0.0; /* number searched */
static double   combi = 1.0; /* number of side chain combos */
static int      totnout = 0, nout = 0; /* number of products */
static int      *Good_Products = 0; /* product bit set */
static int      *Dead_Products = 0; /* forbidden product bit set */
static int      nR;     /* number of R positions (usually 2) */
static int      *nX;    /* number of product dimensions */
static int      *Xct;   /* used for indexing over all products */
static int      **Xsize; /* bytes per field */
static unsigned char    ****X = 0;   /* csln field (F x R x nX )*/
static unsigned char    ***Xin;  /* target fields */
static unsigned char    ***Y;    /* csln core fields (F x R ) */
static unsigned char    ***Yin;  /* target core fields */
static double   ***X2Y;  /* distances between X and X' */
static int      nSym,   /* number of symmetries in this core */
                *CoreSyms, /* flags for all matching core symmetries */
                **SymList; /* symmetry mappings */

static int      DefaultSym[9] = {0, 1, 2, 3, 4, 5, 6, 7, 8};
static int      ReverseSym[2] = {1, 0};
static int      AppendToOutputFile = 0;
static int      NoMorehitsPlease = 0;
static int      UserAborted;
static int      NoCore = 0;
static int      AllCores = 0;
static int      CoreOK = 0;
static int      CoreIsSame = 0;
static int      SideChainOnly = 0;
static int      SideChainsAreSame = 0;
static int      NotBitOutput = 0;
```

```
static char                    comline[2048];
static struct ParseOptions Options[] = {
        {"master",      ParseOptString,         &MasterFile,
             "Prefix for all input files" },
        {"bitset",      ParseOptString,         &BitsetFile,
             "Name is the file with bitset records" },
        {"distance",    ParseOptDouble,         &Distance,
             "Field similarity threshold (default 90.0)" },
        {"cooweight",   ParseOptDouble,         &DWeight,
             "Core coord wt, relative to fields (default )" },
        {"index",       ParseOptInt,            &MasterRecord,
             "Which MasterRecord entry 1-n" },
        {"maxhits",     ParseOptInt,            &NoMorehitsPlease,
             "Maximum number of hits before stopping" },
        {"nocore",      ParseOptInt,            &NoCore,
             "Use -nocore to override inclusion of the core differences" },
        {"allcores",    ParseOptInt,            &AllCores,
             "Use -allcores to search all cores provided" },
        {"input",       ParseOptString,         &InputSource,
             "File from which queries will be read( default stdin). "},
        {"output",      ParseOptString,         &OutputFileBase,
             "File to which hit info will be written. "},
        {"notbits",     ParseOptInt,            &NotBitOutput,
             "Use notbits to output as index ASCII instead of std bitset." },
        {"reaction",    ParseOptString,         &ReactionNeeded,
             "Reaction class for topomer search. "},
        {"details",     ParseOptString,         &ScratchDetails,
             "Details further discriminating the reaction class. "},
        {"sidechain",   ParseOptInt,            &SideChainOnly,
             "Use sidechain to search for similiarity in a single sidechain only. "},
        {"fieldtypes",  ParseOptString,         &FieldTypes,
             "Names of all field types (optional prefix =weight), space separated. Does
    CTOPS if none provided."},
```

```
        {"xweights",    ParseOptString,         &XWeights,
                "Weights of varying sites. Must be nR(+core?) individual weights present (if
    any)."},
    };
5   int UBS_OUTPUT_MESSAGE() { return 0; }    /* just for compiling OK */
    int UIMS2_WRITE_PHOTO() { return 0; }
    int lowercase (s) char *s; {while (*s) { if isupper(*s) *s = tolower(*s); s++;}}
    static int ParseArguments( argc, argv )
    /*+I
10   *
     * This function parses the command line arguments.
     *
     * Returns:  1 on a successful command line parse, 0 otherwise.
     *
15   * Warnings:
     *
     * Errors:
     *
     * Author        Date            Description
20   * ======        ========        ===========
     * G. B. Smith  02-09-93        Original Version
     *
     */
    int     argc;
25  char    **argv;
    {
            int     nargs,
                    noptions = sizeof( Options )/sizeof(Options[0]);
            nargs = UTL_PARSE_OPT( argc, argv, noptions, Options );
30          if( !nargs ) goto SyntaxError;
            return 1;
        SyntaxError:
             fprintf( stderr, "Bad command line argument(s)\n" );
```

```
        return 0;
    }
    static int OpenOutputFile()
    /*+I
     *
     * Returns:  1 on sucesss, else 0
     *
     */
    {
        char    *msg;
        FILE    *fp;
        OutputFile = stdout;
        if( OutputFileBase)
        {
            MakeOutputFileName();
    /*
    ** We need to create output files under the ownership of the REAL user not the
    ** EFFECTIVE user.  This only applies if setuid options are activated.
    */
    {
    struct stat statBuff ;
    int    uid ;
    int    euid ;
           uid = getuid() ;
           euid = geteuid();
        stat(OutputFileName, &statBuff);
     /*
     ** There are two cases
     ** (1) the file to output to exists
     **    Use the ownership of the current owner of the file or if you cant do that
     **    do not do anything.
     ** (2) The file is being created.
     **    use the ownership of the REAL user.
```

```
    */
        if ( access(OutputFileName, F_OK) == 0 )
        { /* If the file exist and the real user is the owner of the file */
            if ( statBuff.st_uid == uid )
                seteuid(uid);
        }
        else
        { /* Create the file as the REAL user */
            seteuid(uid);
        }
    }
        OutputFile = fopen( OutputFileName, (AppendToOutputFile?"a":"wb"));
        if( !OutputFile ) {
            fprintf(stderr,"Error: Failed to open output file \"%s\"\n",
                OutputFileName );
            goto ErrorReturn;
        }
    }
        return 1;
ErrorReturn:
        return 0;
}
static int WhatsTheDifference()
/* builds distance lookup table and initializes default symmetry data structure */
{
 int i, j;
#define pow2(a) ( (a) * (a) )
/* the assignment of codes is based on the following (from gen_pls.c):
  static fpt cutoff[16] = {9999.,  0.,  2.,  4.,  6.,  8., 10., 12.,
                          14., 16., 18., 20., 22., 24., 26., 30. };
*/
 boundary[0] = 9999.; /* missing data ought never to occur. */
 boundary[1] = -0.1 ;
```

```
    for (i=2;i< 15;i++)
      boundary[i] = 2*i-3;
    boundary[15] = 30.0;  /* this is a steep curve with a cutoff at 30! */
    for (i=0;i<16;i++) for (j=0;j<16;j++)
5     Dist[i][j] = pow2( boundary[i] - boundary[j]);
    Distance *= Distance;  /* want to test D^2 directly */
    DWeight *= DWeight;
   /* allocate once for all conceivable symmetry reorderings */
    if (!(SymList = (int **) UTL_MEM_ALLOC( sizeof( int *) * nR * (nR - 1) / 2) ))
10              return 0;
    if (!(CoreSyms = (int *) UTL_MEM_ALLOC( sizeof( int ) * nR * (nR - 1) / 2) ))
                return 0;
    SymList[ 0 ] = DefaultSym;
    SymList[ 1 ] = ReverseSym;
15  return 1;
   }
   static int ReadAField( hex, index, pXP )
   /* converts field from external (ASCii hex) format to internal */
   char *hex;
20 int *index;
   unsigned char **pXP;
   {
      int words, hold;
      char next2[10], *nxhx;
25    words = strlen( hex ) / 2;  /* assuming 8-bit bytes */
      if (! *index ) *index = words;
      if ( words != *index ) {
     /* bad field (most likely NULL), continue anyway */
          *pXP = (unsigned char *) NULL;
30        return 1;
       }
       if (!(*pXP = (unsigned char *) UTL_MEM_ALLOC(words) )) return 0;
       for (words=0, nxhx = hex; words< *index ; words++) {
```

```
        memcpy(next2, nxhx, 2);
        nxhx += 2;
        sscanf( next2, "%2x", &hold );
        *(*pXP + words) = (unsigned char) hold;
    }
    return 1;
}
static int RetrieveInput() {
/* reads the search pattern fields (generated by SYBYL script) */
 int index, R, F;
 char *line;
 double atof();
 if (!InputSource) InputSourceFile = stdin;
  else if (!(InputSourceFile = fopen( InputSource, "r" ) )) {
        fprintf( stdout, "Could not open -input file %s\n", InputSource );
        return 0;
 }
 if (!(Yin = (unsigned char ***) UTL_MEM_ALLOC( sizeof( unsigned char **) * nFType
))
        return 0;
 for (F = 0; F < nFType; F++) {
  if (!(Yin[ F ] = (unsigned char **) UTL_MEM_ALLOC( sizeof( unsigned char *) * nR
))
        return 0;
   memset( Yin[F], 0, sizeof( unsigned char *) * nR );
 }
 if (!NoCore) {
 /* field types are paired closest! */
  for (index = 0; index < nR; index++) for (F = 0; F < nFType; F++) {
 /* a Field is on a single line, no parsing needed */
        if (-1 == UTL_SCAN_GETS( InputSourceFile, "\\", "#", &line))
              return 0;
        if (!ReadAField( line, Xsize[ F ] + index, Yin[ F ] + index )) return 0;
```

```
    }
    for (index = 0; index < 6; index++) {
        if (-1 == UTL_SCAN_GETS( InputSourceFile, "\\", "#", &line))
            return 0;
        CXcoords[ index ] = atof( line );
    }
    if (!(Xin = (unsigned char ***) UTL_MEM_ALLOC( sizeof( unsigned char **) * nFType
    )))
        return 0;
    for (F = 0; F < nFType; F++) {
      if (!(Xin[F] = (unsigned char **) UTL_MEM_ALLOC( sizeof( unsigned char *) * nR
    )))
        return 0;
      memset( Xin[F], 0, sizeof( unsigned char *) * nR );
    }
    for (index = 0; index < nR; index++) for (F = 0; F < nFType; F++ ) {
    /* a Field is on a single line, no parsing needed */
        if (-1 == UTL_SCAN_GETS( InputSourceFile, "\\", "#", &line))
            return 0;
        if (!ReadAField( line, Xsize[ F ] + index, Xin[ F ] + index )) return 0;
    }
    }
    fclose( InputSourceFile );
    return 1;
    }
    static int InitCore() {
    /* readies core file and its input arrays */
      int R, i, F;
      char *foo;

      if (! (CoreFile_File = fopen(Corefile,"r"))) {
          fprintf( stderr, "%s Core file not found.\n", Corefile );
          return 0;
```

```
}
i=0;
while ( i < StartCore )
{
 if ( -1 == UTL_SCAN_GETS( CoreFile_File, "\\", "#", &foo)) return 0;
 if (AllCores) break;
 i++;
}
CoreNow = UTL_STR_SAVE( foo );
/* initialize core data structures */
if (!(Y = (unsigned char ***) UTL_MEM_ALLOC( sizeof( unsigned char **) *
nFType)) )
            return 0;
 for (F = 0; F < nFType; F++) {
  if (!(Y[F] = (unsigned char **) UTL_MEM_ALLOC( sizeof( unsigned char *) * nR)) )

            return 0;
   for (R = 0; R < nR; R++)
    if (!( *( (Y[F]) + R ) = (unsigned char *) UTL_MEM_ALLOC( sizeof( unsigned
char )
             * (*Xsize[F ]) + R ) )) return 0;
 }
 return 1;
}
int CountLines()
{
 int i;
 char *foo;
 /* note that CountLines returns one less than the actual number */
 i=0;
 while ( -1 != UTL_SCAN_GETS( InputSourceFile, "\\", "#", &foo)) i++;
 rewind(InputSourceFile);
 return i;
```

```
}
static int initXarrays ()
{
        int F, i;
        if (!(Xfile = (char **) UTL_MEM_ALLOC( sizeof( char* ) * nR ))) return 0;
        if (!(Xname = (char **) UTL_MEM_ALLOC( sizeof( char* ) * nR ))) return 0;
        if (!(nX = (int*) UTL_MEM_ALLOC( sizeof( int ) * nR ))) return 0;
        if (!(Xct = (int*) UTL_MEM_ALLOC( sizeof( int ) * nR ))) return 0;
        for (i = 0; i < nR; i++) { Xfile[i] = 0; Xname[i] = 0; nX[i] = 0; Xct[i] = 0; }
        if (!(X = (unsigned char ****) UTL_MEM_ALLOC( sizeof( unsigned char ***) *
nFType)) )
                return 0;
        for (F = 0; F < nFType; F++) {
          if (!(X[F] = (unsigned char ***) UTL_MEM_ALLOC( sizeof( unsigned char **)
* nR)) )
                return 0;
          memset( X[F], 0, sizeof( unsigned char **) * nR );
        }
        if (!(Xsize = (int **) UTL_MEM_ALLOC( sizeof( int * ) * nFType ))) return 0;
        for (F = 0; F < nFType; F++) {
          if (!(Xsize[F] = (int *) UTL_MEM_ALLOC( sizeof( int ) * nR ))) return 0;
          for (i = 0; i < nR; i++) *(Xsize[F] + i) = 0;
        }
        return 1;
}
static int initXfiles( i, SideChainsAreSame )
/* reads X file data (reactant descriptors from 2nd comment line of X file ) */
int i, *SideChainsAreSame;
{
        char *foo, *pch;
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if (Xfile[i]) {
/* if this X file is same as last, nothing to do */
```

```
            if (!strcmp( Xfile[ i ], foo ) ) return 1;
            *SideChainsAreSame = FALSE;
            UTL_MEM_FREE( Xfile[i] );
          }
          Xfile[ i ] = UTL_STR_SAVE(foo);
          if (! (InputSourceFile = fopen(Xfile[i],"r"))) {
                fprintf( stdout, "Could not open variation file %s\n", Xfile[i] );
                return 0;
          }
/* reading COMMENT lines to get USER_NAME value for matching */
          if ( -1 == UTL_SCAN_GETS( InputSourceFile, "\\", "", &foo)) return 0;
          if ( -1 == UTL_SCAN_GETS( InputSourceFile, "\\", "", &foo)) return 0;
          if (Xname[i]) UTL_MEM_FREE( Xname[i] );
          Xname[i] = 0;
          pch = strstr( foo, "USER_NAME=" );
          pch += strlen( "USER_NAME=" );
          if (!(Xname[i] = UTL_STR_SAVE( pch ) )) return 0;
          fclose( InputSourceFile );
          return 1;
}
int StartFromBitset()
{
    void *CS_PRDCT_BITSET_OPEN();
    if ( !( bitset =  CS_PRDCT_BITSET_OPEN( BitsetFile, MasterRecord))) return 0;

    if ( !RetrieveMasterFileFromBitset(bitset,
                                       &MasterFile,
                                       &MasterRecord, /*in master file*/
                                       0,
                                       0,
                                       0,
                                       0,
                                       0,
```

```
                                        0,
                                        0,
                                        0,
                                        0,
                                        0,
                                        0,
                                        0,
                                        0,
                                        0,
                                        0,
                                        0,
                                        0   ) ) return 0;

    return 1;
}
/*    1/7/97 DEP: allow reading of bitsets. Since the masterfile must be
                read in any case, the bitset only generates "Dead_Products" */
int InitMasterFile()
/* Read the master file record which is requested;
        failure if it does not match the input line info */
{
  int i, d, size, rxMatch, irx, ns, *Sym;
  char *foo;
  int *fooi;
  if (BitsetFile && ! StartFromBitset()) return 0;
  if (! (MasterFile_File = fopen(MasterFile,"r"))) {
        fprintf( stdout, "%s (master file) not found.\n", MasterFile );
        return 0;
  }
  rxMatch = irx = 0;
  while ( !rxMatch) {
    if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
    if ( strstr(foo,"Reaction class ")) {
        irx++;
```

```
        if (bitset && irx > MasterRecord) return 0; /* the right record did not match */
    /* preliminary match if (1) Reaction Needed matches and (2)
            NO_core must be present if NoCore is TRUE (or vice versa) */
        rxMatch = ( irx >= MasterRecord && strstr( foo, ReactionNeeded )
5               && ((!NoCore && !strstr( foo, "NO_core" ) )
                || ( NoCore &&  strstr( foo, "NO_core" ) ) ) );
    }
    /* if preliminary match, check rest of .mf record  -- first # reactants */
    if (rxMatch) {
10  /* skip name, record / compare number of reagents */
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if ( ! UTL_STR_ATOI(foo, &d) )                  return 0;
        if (!nR) {
15              if (SideChainOnly && d != 1) {
                    fprintf( stdout, "Side Chain only but .mf file references more than
    one side chain.\n" );
                    return 0;
                }
20              nR = d;
                if (!initXarrays()) return 0;
            }
            rxMatch = nR == d;
        }
25      if (rxMatch) {
    /* skip fgpt stuff, record core and side chain file stuff */
            if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
            if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
            if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
30          if (Corefile) UTL_MEM_FREE( Corefile );
            Corefile = UTL_STR_SAVE(foo);
            if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
            if ( ! UTL_STR_ATOI(foo, &StartCore     ) )          return 0;
```

```
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        for (i = 0; i < nR; i++) if (!initXfiles( i, &SideChainsAreSame ) ) return 0;
      }
    } /* read .mf file until we have a matching reaction */
    return 1;
  }
  static int ReadXs() {
  /* reads all topmer fields from all current Xn files */
    int R, F, i, n, ns, realloc, Fd;
    char *CTOPS, *line, *fptr;
    double *dp, **sdptr;
    unsigned char **uc;
    combi = 1.0;
  /* skip the following lengthy stuff if side chains are all the same */
    if (SideChainsAreSame && X[0]) return 1;
    for (R = 0; R < nR; R++) {
      if (! (InputSourceFile = fopen(Xfile[R],"r"))) return 0;
      n = CountLines();
      realloc = n != nX[ R ];
      combi *= (double) n;
      if (realloc && nX[ R ] )
        for (F = 0; F < nFType; F++) {
          for (i = 0; i < nX[R]; i++) UTL_MEM_FREE( *(X[F] + R) + i );
          UTL_MEM_FREE( X[F] + R );
        }

      nX[ R ] = n;
      if (realloc) for (F = 0; F < nFType; F++)
          if (!(*(X[F] + R) = (unsigned char **)
              UTL_MEM_ALLOC( sizeof( unsigned char *) * nX[R]) )) return 0;
    /* starts reading at line 2! */
      for (i = 0; i < nX[R]; i++) {
```

```
        if (-1 == UTL_SCAN_GETS( InputSourceFile, "\\", "#", &line))
            goto error;
    /* generate info for left-to-right read */
        for (F = 0; F < nFType; F++) FOrder[ F ] = strstr( line, FTypes[ F ] );
        do {
          for (Fd = -1, F = 0, fptr = 0; F < nFType; F++)
              if (FOrder[F] && (!fptr || FOrder[F] < fptr)) {fptr = FOrder[F]; Fd =
    F;}
          if (fptr) {
              fptr += strlen( FTypes[ Fd ] ) + 1; /*skipping "CTOPS=" */
              UTL_SCAN_TOKENIZE(fptr,';','\\');
              UTL_SCAN_TOKENIZE(fptr,'>','\\');
              if (!ReadAField( fptr, Xsize[ Fd ] + R, *(X[Fd] + R) + i )) goto error;
              FOrder[ Fd ] = 0;
           }
         } while (fptr);
      }
      fclose( InputSourceFile );
    /* set up X - Y distance vectors */
      if (realloc) for (F = 0; F < nFType; F++) for (ns = 0; ns < nSym; ns++) {
          sdptr = X2Y[ns];
          if (sdptr[R]) UTL_MEM_FREE( sdptr[R] );
          if (!( sdptr[ R ] = (double *) UTL_MEM_ALLOC( sizeof( double ) * nX[R] ) ))
    return 0;
          for (i = 0, dp = sdptr[R]; i < nX[R]; i++) *dp++ = -1.0;
       }
      }
      return 1;
      error:
      fprintf( stdout, "topsim failed reading line %d of %s.\nLast line read was %s.\n",
          i, Xfile[R], line );
      return 0;
    }
```

```
char **ParseQuotedString( SDetails, nDetail, Weights )
char *SDetails;
int *nDetail;
double **Weights;
{
  char *pch, **ppch, *wch, **Details;
  int i;
  double *wt;
 /* first trim string to remove leading/trailing spaces and quotes */
    while (*SDetails == '"' || *SDetails == ' ') SDetails++;
    pch = SDetails + strlen( SDetails ) - 1;
    while (*pch == '"' || *pch == ' ') *pch-- = '\0';
 /* each space is token delimiter  */
    for (i = 0, pch = SDetails; *pch; pch++)
       if (*pch == ' ') i++;
    *nDetail = i+1;
    if (!(Details = (char **) UTL_MEM_ALLOC( sizeof( char * ) * (*nDetail) ) ))
            return 0;
    if (Weights) {
       if (!(*Weights = (double *) UTL_MEM_ALLOC( sizeof( double ) * (*nDetail) ) ))
  return 0;
       wt = *Weights;
    }
    pch = SDetails;
    if (*pch == '"') pch++;
    for (i = 0, ppch = Details; i < *nDetail; i++, ppch++) {
       UTL_SCAN_TOKENIZE(pch,' ','\\');
       *ppch = UTL_STR_SAVE( pch );
       if (Weights) {
 /* note, the copy is now being modified */
          if ((wch = strstr( *ppch, "=" )) ) {
             if (!isweight( wch + 1 )) return FALSE;
             *wt = atof( wch + 1 );
```

```
            *wch = '\0';
          }
          else *wt = 1.0;
          wt++;
        }
        pch += strlen( pch ) + 1;
      }
      return( Details );
    }
    int isweight( s )
    /* returns true if value is a positive decimal value */
    char *s ;
    {
      char *c;
      for (c = s; *c; c++) if (!isdigit( *c ) && ( *c != '.' )) {
        fprintf( stdout, "Bad weight value: %s. Aborting.\n", s );
        return( FALSE );
      }
      return( TRUE );
    }
    int ParseRxn()
    /* parses complex input descriptions */
    {
      char **ParseQuotedString(), **scratch;
      int nRW, i, nX;
      double wtsum;
     /* parse field type information or set up standard (steric) type only */
      if (FieldTypes) {
        if (!(FTypes = ParseQuotedString( FieldTypes, &nFType, &FWeights ) )) return 0;
    /* scale to average weight of unity */
        for (i = 0, wtsum = 0.0; i < nFType; i++) wtsum += FWeights[i];
        wtsum /= (double) nFType;
        for (i = 0; i < nFType; i++) FWeights[ i ] /= wtsum;
```

```
    }
    else {
      nFType = 1;
      if (!( FTypes = (char **) UTL_MEM_ALLOC( sizeof( char * ) ) )) return 0;
      if (!( *FTypes = UTL_STR_SAVE( "CTOPS" ) )) return 0;
      if (!( FWeights = (double *) UTL_MEM_ALLOC( sizeof( double ) ) )) return 0;
      *FWeights = 1.0;
    }
    if (!(FOrder = (char **) UTL_MEM_ALLOC( sizeof(char *) * nFType ) )) return 0;
/* parse any reaction type information present */
    nR = 0;
    if (SideChainOnly) {
        NoCore = TRUE;
        return 1;
    }
    if (!ReactionNeeded) return 0;
    if (ScratchDetails) {
      if (!(ReactionDetails = ParseQuotedString( ScratchDetails, &nDetail, NIL ) )) return 0;
      nR = nDetail;
      if (!initXarrays()) return 0;
    }
    if (!(FROrder = (char **) UTL_MEM_ALLOC( sizeof(char *) * nFType * nR ) )) return
0;
/* parse any user-provided variation weighting */
    CoreWeight = 1.0;
    if (!( RWeights = (double *) UTL_MEM_ALLOC( sizeof( double ) * nR ) )) return 0;
    if (XWeights) {
        if (!(scratch = ParseQuotedString( XWeights, &nRW, NIL ) )) return 0;
/* scratch will just be unfreed memory */
        nX = nR + (NoCore ? 0 : 1);
        if (nRW != nX ) {
            fprintf( stdout, "Mismatch between count of xweights (%d) and needed
(%d).\n", nRW, nX );
```

```
            return 0;
        }
        for (i = 0, wtsum = 0.0; i < nR; i++) if (!isweight( scratch[ i ] )) return
FALSE;
        else {
            RWeights[ i ] = atof( scratch[ i ] );
            wtsum += RWeights[ i ];
        }
        if (!NoCore) if (!isweight( scratch[ nR ])) return FALSE;
        else {
            CoreWeight = atof( scratch[ nR ] );
            wtsum += CoreWeight;
        }
        wtsum /= (double) nX;
        for (i = 0; i < nR; i++ ) RWeights[ i ] /= wtsum;
        if (!NoCore) CoreWeight /= wtsum;
    }
    else for (i = 0; i < nR; i++) RWeights[ i ] = 1.0;
    return 1;
}
int ReadEverything()
{
    if (!MasterFile && !BitsetFile) return 0;
    if (!ParseRxn()) return 0;
    setbits_nbits_Init();
    if (!InitMasterFile() ) return 0;
    if (!InitCore() ) return 0;
    if (!WhatsTheDifference()) return 0;
    if (!RetrieveInput() ) return 0;
    return 1;
}
static int InitSym( nsym )
int nsym;
```

```
{
/* sets up symmetries to consider as described for core
   ONLY 2 reactants considered for now!
   assumes that CoreNow is pointing to the appropriate structure */
5    int i, F, maxsym;
   double **x2y;
/* get symmetry from current core molecule if not supplied by caller */
   nSym = nsym;
   if ( !nSym ) {
10      if ((!strstr( CoreNow, "SYM=" )) || (strstr(CoreNow, "SYM=0")) ) nSym = 1;
      if (strstr(CoreNow, "SYM=1")) nSym = 2;
/* add more categories here */
   }
      for (i = 0; i<nSym; i++) CoreSyms[ i ] = 1;
15 /* allocate distance arrays to max possible for nR */
   if (!X2Y) {
     for (maxsym = 1, i = 0; i < nR; i++) maxsym *= (i+1);
     if (!(X2Y = (double ***) UTL_MEM_ALLOC( sizeof( double **) * nFType ) ))
  return 0;
20      for (i = 0; i < maxsym; i++) {
         if (!(X2Y[i] = (double**) UTL_MEM_ALLOC( sizeof( double *) * nR) )) return 0;
          memset( X2Y[i], 0, sizeof( double *) * nR );
       }
     }
25    return nSym;
  }
  int ReadCoreTopomers( CoreOK )
  int *CoreOK;
  {
30 /* returns 1 unless fatal error. Sets CoreOK to TRUE if this mf entry is OK
     Also sets up symmetry considerations (which are core structure dependent).
     assumes that CoreNow is pointing to the appropriate structure */
   int foo, i, R, F, Fd, Rd, rf, skipcore, ns, *Sym;
```

```
char label[15], *nxTop, *cstart, *fptr;
char *cnames[] = {"NX=","NY=","NZ=","CX=","CY=","CZ="};
double coo;
double atof();
skipcore = NoCore;
/* always consider both matches iff no core */
if(skipcore) InitSym( nR );
      else skipcore = ! InitSym(0);
/* check for any symmetry-allowed rxn by rxn match of all reactant name "details" */
for (ns = 0; ns < nSym; ns++) if (CoreSyms[ns]) {
      Sym = *(SymList + ns);
      *CoreOK = TRUE;
      if (!SideChainOnly)
        for (i = 0; i < nR && *CoreOK; i++)
         if (!strstr( ReactionDetails[ Sym[ i ] ], Xname[ i ] ))
              *CoreOK = FALSE;
       if (*CoreOK) break;
 }
 if (skipcore || CoreIsSame || !(*CoreOK )) return 1;
 nxTop = CoreNow;
/* read left-to-right, so record all starting points;
  assume that coords are bunched and appear only once
*/
 for (F = 0; F < nFType; F++) for (R = 0; R < nR; R++) {
    sprintf( label, "%s%d", FTypes[ F ], R + 1 );
    if (!( FROrder[ F * nR + R ] = strstr( nxTop, label ) ) ) {
/* some requested datum missing; then this core entry has no topomer data; use it */
        *CoreOK = 0;
        return 1;
    }
 }
 cstart = strstr( nxTop, cnames[ 0 ] );
 do {
```

```
/* find next datum in left-to-right order */
   for (F = 0, fptr = 0; F < nFType; F++) for (R = 0; R < nR; R++) {
      rf = F * nR + R;
      if (FROrder[rf] && (!fptr || FROrder[rf] < fptr)) {fptr = FROrder[rf]; Fd = F;
5  Rd = R;}
   }
   if (cstart && (!fptr || cstart < fptr)) {fptr = cstart; Fd = -1; }
   if (fptr) {
/* unpack next piece of data to proper location */
10       if (Fd >= 0) {
/* then datum is a field */
            fptr += strlen( FTypes[ Fd ] ) + 2; /*skipping "CTOPn=" */
            UTL_SCAN_TOKENIZE(fptr,';','\\');
            UTL_SCAN_TOKENIZE(fptr,'>','\\');
15          if (!ReadAField( fptr, Xsize[ Fd ] + Rd, Y[Fd] + Rd )) return 0;
            FROrder[ Fd * nR + Rd ] = 0;
         }
         else {
          for (i = 0; i < 6; i++) {
20  /* the next data are coordinates */
    /* read coords, save as distances squared */
            cstart = strstr( cstart, cnames[i]);
            if (!cstart) {
     /* then this core entry has no topomer data */
25             *CoreOK = 0;
               return 1;
            }
            cstart += strlen(cnames[i]);
            UTL_SCAN_TOKENIZE(cstart,';','\\');
30          UTL_SCAN_TOKENIZE(cstart,'>','\\');
            coo = CXcoords[ i ] - atof(cstart);
            CXdiffsq[ i ] = coo * coo * DWeight;
            cstart += strlen( cstart ) + 1;
```

```
            }
            cstart = 0;
           }
         }
5    } while (fptr);
     return 1;
     }
     int CoreMatches( CoreOK )
     int *CoreOK;
10   {
     /* returns 1 unless fatal error. Sets CoreOK to FALSE if no compound having
            this core can possibly match */
      int F, R, i, ns, *Xct, ct;
      double sqrt(), totd, xount, cdiff;
15    unsigned char *ptr, *qtr;
     if (NoCore || CoreIsSame) {
            *CoreOK = TRUE;
            return 1;
     }
20   /* can check for coordinate discrepancy fast! */
      for (i = 0, cdiff = 0.0; i < 6; i++) cdiff += CXdiffsq[i];
      if (cdiff > Distance) {
            *CoreOK = FALSE;
            return 1;
25    }
      for (F = 0, totd = cdiff; F < nFType; F++) for (R = 0; R < nR; R++) {
            if (totd > Distance) break;
            ptr = (unsigned char *) *(Y[ F ] + R);
            qtr = (unsigned char *) *(Yin[F] + R);
30          if (!ptr || !qtr) xount = 999999.0;
            else for(xount=0.0, i=0; i < *(Xsize[F] + R); i++, ptr++, qtr++)
              xount += Dist[  *ptr & 0x0F      ][  *qtr & 0x0F      ]
                    + Dist[ (*ptr & 0xF0) >> 4][ (*qtr & 0xF0) >> 4] ;
```

```
        totd += xount * FWeights[ F ] / (double) nR;
    }
    CoreDistance = totd * CoreWeight;
    *CoreOK = totd <= Distance;
    return 1;
}
int FindXMatches () {
  int R, F, i, ns, ct, *Sym, size,  what ;
  double totd, d, **sdptr, *dptr, xount;
  unsigned char  *ptr, *qtr;
/* reinitialize indices for permuting over all products  --
      code is general for any number of variable positions */
  for (i = 0; i < nR; i++) Xct[i] = 0;
              AddressSize(nR, nX, &size);
              size = (size + 31 )/32 * 4;
  if (bitset) /* assumes actuallsizes matches current sizes!*/
            {
              if (!(Dead_Products = (int *) UTL_MEM_ALLOC(size))) return 0;
              CS_PRDCT_BITSET_TO_RAW( bitset, Dead_Products,  0);
              not_here(Dead_Products,size  );
            }

  while ( TRUE ) {
/* exit elsewhere when all products are enumerated */
        IndexesToAddress( nR, nX, &what, Xct);
      if (Dead_Products &&
         TestDead(0, what) ) goto tupledone;  /* not doing this one! */

      for (ns = 0; ns < nSym; ns++) if (CoreSyms[ns]) {
/* process all symmetries of current side chain combo */
        Sym = *(SymList + ns);
        sdptr = *(X2Y + ns);
        for (R = 0, totd = CoreDistance;  R < nR; R++) {
```

```
            if (totd > Distance) break;
    /* compute next distance if not already done -- DEP knows how this works! */
            dptr = (*(sdptr + R )+Xct[R]);
            if ((*dptr) < 0.0) for (F = 0; F < nFType; F++ ) {
              ptr = (unsigned char *) *( *(X[F] + R) + Xct[ R ]);
              qtr = (unsigned char *)     *(Xin[ F ] + Sym[ R ]) ;
              if (!ptr || !qtr) {*dptr = 999999.0; break;}
              else {
                for(xount=0.0, i=0; i < *(Xsize[F] + R); i++, ptr++, qtr++)
                 xount += Dist[  *ptr & 0x0F      ][  *qtr & 0x0F      ]
                       + Dist[ (*ptr & 0xF0) >> 4][ (*qtr & 0xF0) >> 4]  ;
                *dptr += xount * FWeights[ F ];
               }
             }
            totd += *dptr * RWeights[ R ];
           }
      /* if hit, write it out */
           if (totd <= Distance) {
            if (NotBitOutput || nR != 2) {
    /* ASCII index form of output  -- also REQUIRED if more than 2 varying elements */
              if (!OutputFile && !OpenOutputFile() ) return 0;
              for (R = 0; R < nR; R++) fprintf( OutputFile, "%6d ", Xct[R] + 1 );
              fprintf( OutputFile, "%6d%8.2f%8.2f%8.2f\n", StartCore,
                    sqrt(totd), sqrt(CoreDistance), sqrt(totd - CoreDistance) );
            }
            else {
              if (!Good_Products ) {
                    if (!(Good_Products = (int *) UTL_MEM_ALLOC( size ) )) return
      0;
                    memset( Good_Products, 0, size );
              }
              FlagProduct(Good_Products, 0, 0, what );
            }
```

```
            nout++;
            if (NoMorehitsPlease && nout >= NoMorehitsPlease) goto done;
    /* output only one acceptable symmetry per product */
            goto tupledone;
5           }
          }
    /* generate next index tuple, AKA candidate product */
    tupledone:
          ct = nR - 1;
10        while ( TRUE ) {
            Xct[ ct ] ++;
            if (Xct[ ct ] < nX[ ct ]) break;
     /* finished when first index exceeds limit -- the other exit  */
            if (ct == 0) goto done;
15          Xct[ ct ] = 0;
            ct--;
          }
      }
     done:
20   /* output any products from this dataset */
      if (NotBitOutput || nR != 2) {
        if (OutputFile) fclose(OutputFile);
        OutputFile = 0;
      }
25    else if (Good_Products) {
            WriteStdFile();
            UTL_MEM_FREE( Good_Products );
            Good_Products = (int*) 0;
      }
30    return 1;
     }
     int MakeOutputFileName() {
     /* a run may produce multiple files, and the user probably can't tell,
```

```
        so append a sequence _# to subsequent base names */
    if (!nOutFiles) {
        sprintf( OutputFileName, "%s", OutputFileBase );
/* get base name ready for next call */
        strtok( OutputFileBase, "." );
    }
    else sprintf( OutputFileName, "%s_%d.%s", OutputFileBase,
                nOutFiles, OutputFileBase + strlen(OutputFileBase) + 1 );
    nOutFiles++;
}
int WriteStdFile() {
/* writes out the bit set of products */
int sizes[2];
int allocSizes[2] ;
int numInSites[2] ;
void *compressed ;
int  total ;
        sizes[0] = nX[0] ;
        sizes[1] = nX[1] ;
        numInSites[0] = numInSites[1] = -1  ;
        allocSizes[0] = allocSizes[1] = -1 ;
        compressed = NIL;
        total      =  0;
          MakeOutputFileName();
          WriteOutCheckPointFile( OutputFileName,
                MasterFile,
                MasterRecord,
                comline,
                Good_Products,
                0,
                2,
                sizes,
                allocSizes,
```

```
                nout,
                numInSites,
                total,
                compressed);
}
int ReadNextCore( SideChainsAreSame, CoreIsSame ) .
int *SideChainsAreSame;
int *CoreIsSame;
{
/* continues reading through master file for more matching Reaction Classes.
        If the side chain files have the same name, can skip rebuild of X diffs  */
 char *foo;
 int i, d, rxMatch = 0, val;
 if (AllCores) {
        if ( -1 == UTL_SCAN_GETS( CoreFile_File, "\\", "#", &foo)) fclose(
CoreFile_File );
        else {
/* get next core ready and quit */
         CoreNow = UTL_STR_SAVE(foo);
         *SideChainsAreSame = TRUE;
         StartCore++;
         return 1;
        }
 }
 while ( !rxMatch ) {
  if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
 /* preliminary match if (1) Reaction Needed matches and (2)
          NO_core must be present if NoCore is TRUE (or vice versa) */
    rxMatch = ( strstr(foo,"Reaction class ") && strstr(foo, ReactionNeeded)
            && ((!NoCore && !strstr( foo, "NO_core" ) )
             || ( NoCore &&  strstr( foo, "NO_core" ) ) ) );
    if (feof(MasterFile_File)) return 0;
 /* skip name, record number of reagents */
```

```
    if (rxMatch) {
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if ( ! UTL_STR_ATOI(foo, &val) ) return 0;
        if (val != nR) rxMatch = 0;
    }

    if (rxMatch) {
  /* skip fgpt stuff, record core and side chain file stuff */
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        *CoreIsSame = TRUE;
        if (strcmp( foo, Corefile )) {
          *CoreIsSame = FALSE;
          UTL_MEM_FREE( Corefile );
          Corefile = UTL_STR_SAVE(foo);
        }
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if ( ! UTL_STR_ATOI(foo, &val ) )                return 0;
        if (val != StartCore ) *CoreIsSame = FALSE;
        StartCore = val;
        if (! *CoreIsSame ) {
          if (CoreFile_File) fclose(CoreFile_File);
          if (! (CoreFile_File = fopen(Corefile,"r"))) return 0;
          i=0;
          while ( i < StartCore ) {
              if ( -1 == UTL_SCAN_GETS( InputSourceFile, "\\", "#", &foo)) return 0;
              if (AllCores) break;
              i++;
          }
          CoreNow = UTL_STR_SAVE( foo );
        }
```

```
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        *SideChainsAreSame = TRUE;
        for (i = 0; i < nR; i++) if (!initXfiles( i, SideChainsAreSame ) ) return 0;
    }
   }
   return 1;
  }
  /* this belongs in the utl module, actually */
  int MakeComLine( char *line, int len, int argc, char **argv)
  {
   int i, nch, totch = 0;
   sprintf(line,"%s ",argv[0]);
   for(i=1;i<argc && totch <= len;i++)
   {
    nch = strlen(line);
    line += nch;
    totch += nch;
    if (totch < len ) sprintf(line,"%s ",argv[i]);
   }
  }
  int CheckPointProgram(void) {
   fprintf(stderr,"CheckPointProgram() is a lonely stub in topsim.c!\n");
  }
  int main( argc, argv )
  int    argc;
  char   **argv;
  {
        int processing;
        if( !ParseArguments( argc, argv ) )
              goto SyntaxError;
        MakeComLine( comline, 2048, argc, argv );
        if (!ReadEverything())    goto FailureExit;
```

```
        processing = 1;
        while (processing) {
          if (!ReadCoreTopomers( &CoreOK )) goto FailureExit;
         if (CoreOK && !CoreMatches( &CoreOK )) goto FailureExit;
          if (CoreOK && !ReadXs()) goto FailureExit;
          searched += combi;
          if (CoreOK && !FindXMatches()) goto FailureExit;
          totnout += nout;
          nout = 0;
          processing = ReadNextCore( &SideChainsAreSame, &CoreIsSame ) &&
               (!NoMorehitsPlease || nout < NoMorehitsPlease);
        }
         fprintf(stdout, "Normal Exit: %d of %f are neighbors\n", totnout, searched );
        UserAborted ? exit(ErrorExit) : exit(GoodExit);
SyntaxError:
        exit(1);
FailureExit:
        exit(ErrorExit);
}
/*
  numVariations is number of dimensions Y_01, Y_02 etc (normally 2)
  dsize contains the nY_01, nY_02 etc
  address is the bit number (0 to N-1)
  choices will contain the offsets (0 based) of Y_01, Y_02 etc. on return
*/
int AddressToIndexes(int numVariations, int *allPtr, int address, int *chPtr )
{
        for ( chPtr += (numVariations - 1 ), allPtr += (numVariations - 1) ;
             numVariations-- ;
             allPtr--,  chPtr-- )
        {
             *chPtr = address % *allPtr;
             address = address / *allPtr;
```

```
        }
        return 1;
    }

    int  IndexesToAddress(int numVariations, int *allPtr, int *address, int *ind)
5   {
    int    i ;
    int    indx = 0 ;
           for (i=0;i<numVariations;i++)
             indx += indx * allPtr[i] + ind[i];
10         *address = indx;
           return 1 ;
     }
     int AddressSize(int numVariations, int *allPtr, int *size)
     {
15    for ( *size = 1 ; --numVariations; allPtr++) *size *= *allPtr;
      return 1;
     }

     int not_here( what, nbytes )
     unsigned char *what;
20   int nbytes;
     {
      for ( ; nbytes; --nbytes) *what++ = ~*what;
      return 1;
     }
```

Appendix "T"

```
@macro FragCTOPS ChSp
#
==========================================================
==========================
# Entry point for Web-based topomeric search initialization
#
# sets up a set of topomeric searches, by identifying topomer data arising
from
# substructural searching of SLN patterns found in topfrag.tbl to the
# query structure and generating the topomeric data and search command file
entry
# for all resulting fragmentations of the query structure.
#
#    The Query SLN(s) are assumed to be referenced by $CS_QUERY;
#    The file(s) to be searched are referenced by $CS_DATASET (space
separated)
#    The directory where command files are to be written is $CS_TEMPDIR
#    The GUI parameters are to be in $CS_PARAMETERS
#    The name of the output file(s) is to be in $CS_OUTPUT
# read in the data
globalvar CTOP
globalvar ACD!TopInited
localvar fcmdn fcmd tsln dist t base mf mfo nln nxid ferr ferrn rxids doit
# check the input parameters
    setvar ferrn %cat( $CS_TEMPDIR "/CSerror.log" )
    setvar ferr %open( $ferrn "w" )
    setvar flogn %cat( $CS_TEMPDIR "/topfrag.log" )
    setvar flog %open( $flogn "w" )
    setvar fcmdn %cat( $CS_TEMPDIR "/CSCommands.cmd" )
    setvar fcmd %open( $fcmdn "w" )
    if %not( $fcmd )
        %write( $ferr could not open temp file $fcmdn to write ChemSpace search
```

```
cmds. Quitting ) >$nulldev
        return
      endif
      for tsln in $CS_QUERY
5       if %pos( "." $tsln )
          setvar nogood TRUE
          if %pos( "<" $tsln )
            if %gt( %pos( "." $tsln ) %pos( "<" $tsln ) )
              setvar nogood
10          endif
          endif
          if $nogood
            %write( $ferr Topomeric searches require a monomolecular search target.
    Quitting ) >$nulldev
15          goto error
          endif
        endif
        %write( $flog QUERY: $tsln >$nulldev
        setvar dist %CS_param_parse( distance $CS_PARAMETERS 91.0 )
20      if %not( $dist )
          %write( $ferr No topomeric distance provided. Quitting ) >$nulldev
          goto error
        endif
        setvar priority %CS_param_parse( priority $CS_PARAMETERS 3.0 )
25      if %not( $priority )
          %write( $ferr No reaction priority provided. Quitting ) >$nulldev
          goto error
        endif
        %write( $flog Fragment Priority: $priority ) >$nulldev
30      setvar CTOP[ ONLY1 ] %CS_param_parse( only_subs $CS_PARAMETERS )
        if $CTOP[ ONLY ]
          %write( $flog Matching Side Chain Only ) >$nulldev
        endif
```

```
    setvar CTOP[ WEIGHTS ] %CS_param_parse( xweights $CS_PARAMETERS )
    if $CTOP[ WEIGHTS ]
        %write( $flog User Specified Weighting as: $CTOP[ WEIGHTS ] ) >$nulldev
        for w in $CTOP[ WEIGHTS ]
5           setvar pats %search2d( $tsln %arg( 1 %set_unpack( $w ) ) NoDup 0 y )
            if %not( $pats )
                %write( $ferr Weighted search for fragment %arg( 1 %set_unpack( $w ) )
  not
  in $tsln -- can't happen! ) >$nulldev
10              goto error
            else
             if %gt( %count( $pats ) 1 )
                %write( $flog NOTE: Multiple hits for weighting fragment %arg( 1
  %set_unpack( $w ) ) in $tsln ) >$nulldev
15           endif
            endif
           endfor
       endif
       setvar CTOP[ CHBD ] %CS_param_parse( hbonding $CS_PARAMETERS )
20     if $CTOP[ CHBD ]
           %write( $flog FIELDS include Hydrogen Bonding with weight of $CTOP[ CHBD ]
  )
  >$nulldev
       endif
25     zap m1 >$nulldev
       %sln_to_mol( m1 $tsln ) >$nulldev
       if %molempty( m1 )
           %write( $ferr SYBYL cannot handle search target (SLN is: $tsln ).
  Quitting ) >$nulldev
30         goto error
       endif
       setvar t %mol_info( m1 NATOMS )
       FILLVALENCE M1(*) H 1.0 1.5 1.0 1.5 >$nulldev
```

```
if $CTOP[ ONLY1 ]
    if %neq( %mol_info( m1 NATOMS ) %math( $t + 1 ) )
     %write( $ferr Side chain search but target $tsln has other than one
unfilled valence ) >$nulldev
     goto error
    endif
   else
    if %neq( %mol_info( m1 NATOMS ) $t )
     %write( $ferr Search Target $tsln has unfilled valences. Quitting )
>$nulldev
     goto error
    endif
   endif
  if $CTOP[ ONLY1 ]
# only one side chain to model is a special case
   CTOP!SideChainOnly $fcmd $ferr $flog $dist
  else
 # check for custom topomer fragmentation table or selection
   setvar tftabn
   setvar tfrows
   if $CS_TOPFRAG
    setvar t %pos( "_" $CS_TOPFRAG )
    if %not( $t )
     %write( $ferr Custom table name $CS_TOPFRAG missing an "_" ) >$nulldev
     goto error
    else
     setvar tftabn %substr( $CS_TOPFRAG 1 %math( $t - 1 ) )
     setvar tfrows  %substr( $CS_TOPFRAG %math( $t + 1 ) )
    endif
   endif
   if %set_and( "%set_create( %table_name() )" TOPFRAG )
    table close TOPFRAG
   endif
```

```
    if %not( $tftabn )
        setvar tftabn %cat( $DSERV_TB topfrag.tbl )
    endif
    table recall $tftabn >$nulldev
    if %not( %set_and( "%set_create( %table_name() )" TOPFRAG ) )
        %write( $ferr $tftabn not found. Quitting ) >$nulldev
        goto error
    endif
    %write( $flog Topomer fragmentation table is %cat( $DSERV_TB topfrag.tbl
) ) >$nulldev
# initialize random file name sequence generator
    setvar t %time()
    setvar base %rand( %substr( "$t" %math( %strlen( "$t" ) - 6 ) 2 ) )
    TAILOR SET MAXIMIN2 MAXIMUM_ITERATIONS 1000 | |
    %write( $flog Master file(s): $CS_DATASET ) >$nulldev
    %write( $flog TOPFRAG table: $tftabn  -- Row selection: $tfrows )
>$nulldev
    if %not( $tfrows )
        setvar tfrows %set_create( %range( 1 %table_attribute( NROWS ) ) )
    endif
    for rxid in %set_unpack( $tfrows )
# processing ...
        %write( $flog - - - - - - - - - - - - - - - - - - - - - - ) >$nulldev
# chcek priority
        TABLE Default TOPFRAG
        if %gt( %rcell( $rxid PRIORITY ) $priority )
            %write( $flog TOPFRAG entry $rxid priority > $priority. ) >$nulldev
            break
        endif
        setvar CTOP[RxnCount][$rxid] 0
        if %CS_ReactantMatch( $rxid $fcmd $ferr $tsln $flog )
            %write( $flog > > > Topomer search queueing (TOPFRAG row $rxid) )
>$nulldev
```

```
                CS!Queue_Search $fcmd $rxid $dist $flog
            endif
        endfor
    endif
endfor
# may need to purge or rename error file here!
    %close( $fcmd )
    %close( $ferr )
    %close( $flog )
    return
error:
    %close( $fcmd )
# ensure nothing in search command file !
    %file_delete( $fcmdn ) >$nulldev
```